January 2015

Multi-level Systems Modeling and Optimization

Shreyas Vathul Subramanian
Purdue University

Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Shreyas Vathul Subramanian

Entitled

Multi-level Systems Analysis and Optimization for Novel Aircraft

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Daniel A. DeLaurentis
Chair

William Crossley

Denfeng Sun

Michael Grant

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Daniel A. DeLaurentis

Approved by: Weinong Wayne Chen 10/15/2015

Head of the Departmental Graduate Program Date

MULTI-LEVEL SYSTEMS MODELING AND OPTIMIZATION FOR NOVEL AIRCRAFT

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Shreyas Vathul Subramanian

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2015

Purdue University

West Lafayette, Indiana

To Amma (my strength), and Daddy (my inspiration)

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

ABSTRACT

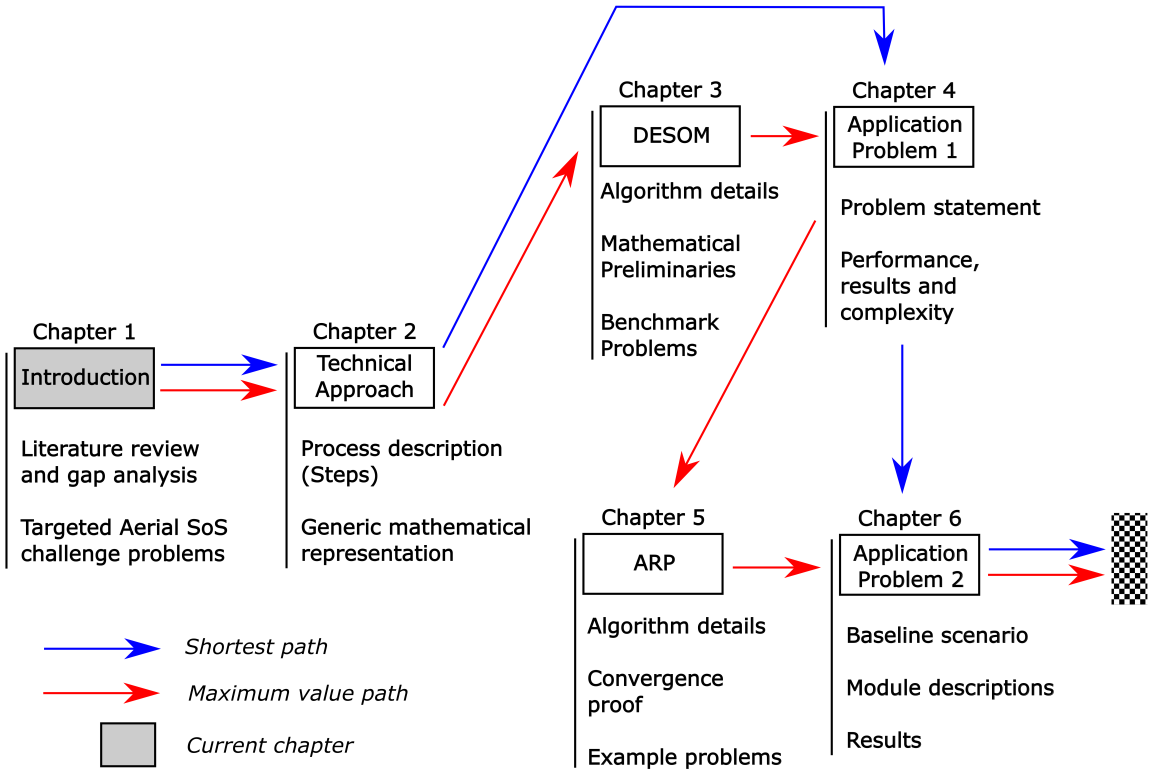
Subramanian, Shreyas Vathul Ph.D., Purdue University, December 2015. Multi-level Systems Modeling and Optimization for Novel Aircraft. Major Professor: Daniel A. DeLaurentis.

This research combines the disciplines of system-of-systems (SoS) modeling, platform-based design, optimization and evolving design spaces to achieve a novel capability for designing solutions to key aeronautical mission challenges. A central innovation in this approach is the confluence of multi-level modeling (from sub-systems to the aircraft system to aeronautical system-of-systems) in a way that coordinates the appropriate problem formulations at each level and enables parametric search in design libraries for solutions that satisfy level-specific objectives. The work here addresses the topic of SoS optimization and discusses problem formulation, solution strategy, the need for new algorithms that address special features of this problem type, and also demonstrates these concepts using two example application problems - a surveillance UAV swarm problem, and the design of noise optimal aircraft and approach procedures.

This topic is critical since most new capabilities in aeronautics will be provided not just by a single air vehicle, but by aeronautical Systems of Systems (SoS). At the same time, many new aircraft concepts are pressing the boundaries of cyber-physical complexity through the myriad of dynamic and adaptive sub-systems that are rising up the TRL (Technology Readiness Level) scale. This compositional approach is envisioned to be active at three levels: validated sub-systems are integrated to form conceptual aircraft, which are further connected with others to perform a challenging mission capability at the SoS level. While these multiple levels represent layers of physical abstraction, each discipline is associated with tools of varying fidelity forming strata of 'analysis abstraction'. Further, the design (composition) will be guided by a suitable hierarchical complexity metric formulated for the management of complexity in both the problem (as part of the generative

procedure and selection of fidelity level) and the product (i.e., is the mission best achieved via a large collection of interacting simple systems, or a relatively few highly capable, complex air vehicles). The area of optimization in evolving design spaces has had only limited exploration, but will be studied and incorporated into the SoS optimization framework. We envision a framework that resembles a multi-level, mult-fidelity, multi-disciplinary assemblage of optimization problems.

The challenge is not simply one of scaling up to a new level (the SoS), but recognizing that the aircraft sub-systems and the integrated vehicle are now intensely cyber-physical, with hardware and software components interacting in complex ways that give rise to new and improved capabilities. The work presented here is a step closer to modeling the information flow that exists in realistic SoS optimization problems between sub-contractors, contractors and the SoS architect.



Readers may follow directions shown above to guide their reading experience. The current chapter will be highlighted in grey as shown. Detailed explanations of two of our algorithms, Differential Evolution with Self Organizing Maps (DESOM) in chapter 3 and Adaptive Random Projection (ARP) in chapter 5 are important, and provide a deeper coverage than that in chapters 4 and 6 demonstrating their use in application problems 1 and 2 respectively.

1. Introduction

Advancement in aircraft design science and optimization is being given considerable attention as manufacturers attempt to reduce the time-to-market of new projects. [1,2] However, more challenging requirements and greater demands for system verification have increased complexity of both, the design artifact and the design process. Aircraft design optimization problems are frequently non-convex, multi-modal and ill-conditioned. This makes the complete Multidisciplinary Design Optimization (MDO) of aircraft systems prohibitively expensive. [1] Analysis and optimization of these complex systems, characterized by such high-dimensional design spaces, require an array of closely knit, discipline-specific modules. While a multitude of relevant disciplines contribute to the 'multi-disciplinary' aspect of MDO, researchers have intelligently made use of 'multi-fidelity' approaches to obtain reliable results under reduced computational effort. [3–6] Typical optimization procedures target specific parts of an aircraft's mission, which then translate to specific technical requirements. For example, certain aspects of a commercial airliner may be optimized for cruise. MDO techniques have been very successfully implemented in the design of a particular system (in this example, an aircraft, that is optimized for cruise). The aircraft itself, however, is composed of several other sub-systems that may be designed and optimized by companies that specialize in *that* product (for example, a Pratt and Whitney aircraft engine). Several existing design procedures are restricted to a single level of analysis that may yield high quality results for a given sub-problem. Therefore, a 'multi-level' synthesis of optimization problems that treats every system and sub-system involved in a mission is required for obtaining realistic results. It is important to recognize that an assemblage of optimal sub-systems may not necessarily result in an optimum system. Extending this idea to SoS, multiple stakeholders, contractors and sub-contractors may contribute towards fulfilling an overall top level capability. Component manufacturers (eg: circuit boards), sub-system manufacturers (eg: multi-spectral camera), system manufacturers (eg: surveil-

lance UAV) and the SoS architect (eg: surveillance swarm) may all have independent design and optimization procedures. Levels of hierarchy presented here are linked through "products" or "objects" designed independently, but all contributing towards a common mission. While there have been several working approaches into the effort of formulating and solving multi-level, multi-fidelity, multi-disciplinary optimization problems, the approaches generally focus on the design and optimization of a single system. The concept of Systems-of-Systems, due to its hierarchical nature with operationally and managerially independent systems designed and manufactured by respective stakeholders, yields itself naturally to the structure of the problem we have begun alluding to in this paragraph. Ayyalasomayajula also points out that the solution methodology for SoS problems depends on the distinctive features of the problem itself. [7]

While an aircraft development effort must be viewed in the context of the SoS within which it exists, [8] the aircraft itself is a complex system composed of a set of interdependent sub-systems. This seemingly clear hierarchy masks the complicated and varied interactions that exist between the levels. These subtle interactions are responsible for the observed heterogeneity of designs that satisfy the same capability. The top-level SoS capability needs to guide the system (aircraft) level synthesis, that in turn drives sub-system (example: avionics) requirements. For example, a swarm of UAVs (SoS level) may provide continuous surveillance (capability) over a predefined area. At this level, communications, patterns and trajectories may need to be optimized for improving performance or reducing the cost of operation. The UAV itself (system level) can be optimized across multiple disciplines such as aerodynamics, structure and propulsion for satisfying the given capability. Individual components (like an infrared camera) that characterize the UAV will also have to be chosen carefully and optimized (sub-system level). In this work, we attempt to solve such problems by directly tying sub-system and system design optimization to SoS-level objectives and requirements, while mimicking the information flow that exists in the realistic design of a mission-capable SoS. In particular, we are interested in Aerial SoS applications.

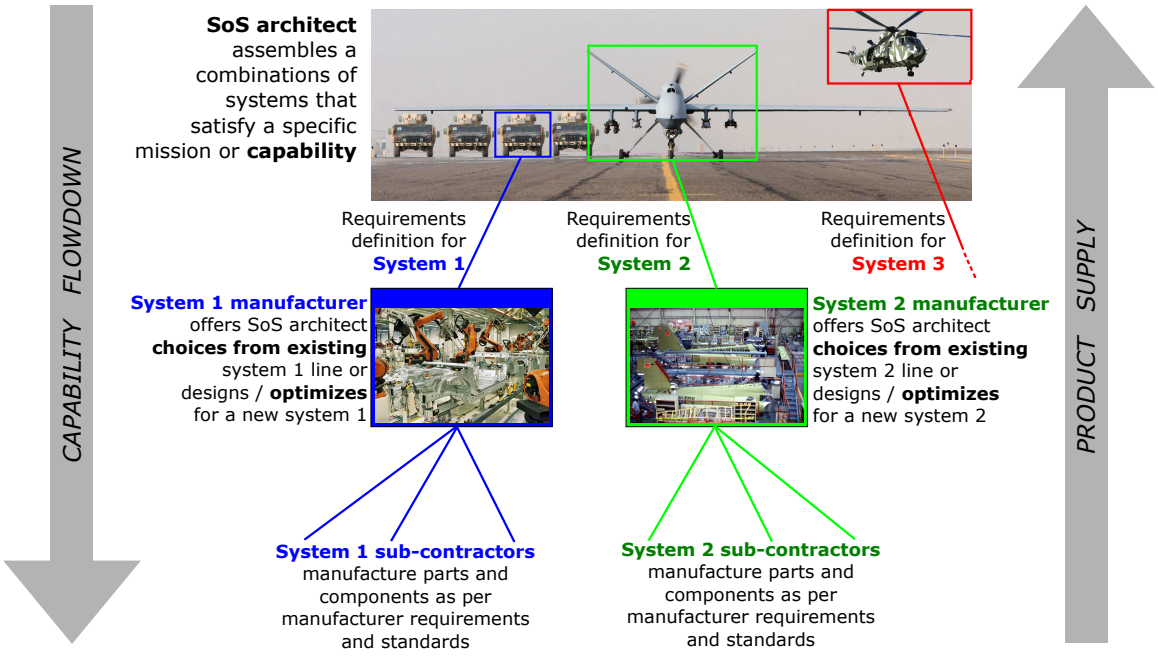


Figure 1.1.: Levels and systems that exist in a sample SoS optimization problem

The infographic in figure 1.1 describes a SoS that is to be designed and optimized for a particular collaborative mission. A group of operationally and managerially independent systemsv that form an SoS are to provide a particular capability, or fulfill a given mission (the actual mission description is not important here). Each of these systems are designed and optimized for a particular requirement provided by the top level (or the SoS level). The systems themselves may be assembled by the primary manufacturer or contractor, with other sub-contractors manufacturing relevant sub-systems needed in the assembly of a system. Thus, the capability to be achieved by the SoS flows down as requirements posed by contractors to sub-contractors (large arrow on the left of figure 1.1). Sub-products are assembled to form products, which are then grouped to form a mission-suitable SoS (large arrow on the right of figure 1.1). It is such problems that shall be considered in this work.

1.1 Literature Review

Customers in the Aerospace industry now ask for broad capabilities and solutions (demonstration and implementation) rather than for specific individual systems. [9] Several authors have showcased important methods that produce high quality solutions at a particular desired level of fidelity or modeling. First, we explore the possibility of using **existing methods**, either directly or with modifications and suggested extensions of the methods[1]. We recognize that the problem structure that we are interested in is special, and therefore, methods that could be used directly (without any modification) may be rare, or may solve specific versions of the problem at hand. We use the vast literature on MDO as a starting point here due to a large and obvious overlap in the problem features that will made clear in the forthcoming sections.

Mane *et al*. handled this special kind of multi-level problem by using an appropriate mixture of current and new systems in the form of a SoS to enable design and optimization. The authors perform aircraft sizing at the system level to achieve a capability of transporting passengers at the SoS level. Note that the framework is active only at two levels. SoS operate in an uncertain environment and evolve with time. The authors in [10] attempt to solve the problem by simplifying it as a static, deterministic case. Taylor and Weck attempted the same problem, but with an integrated approach rather than a decomposed one. However, the parameterization of aircraft in their research was simplified – the SoS architect was allowed to choose one of three fixed aircraft to be used in the network. Hence here too, the problem was a two level, deterministic one. [11]

Other examples of research involving simultaneous airline route structure and aircraft design optimization are [12] and [13]. Nusawardhana and Crossley use a circular sequential decomposition at the core of their framework. Again, the authors significantly simplify the aircraft design portion of the research by having a lower level optimization problem with primarily three aircraft design variables – Thrust-to-weight ratio (T/W), Wing Loading (W/S), and Aspect Ratio (AR). Moreover, the SoS architect is allowed to choose systems

[1] Some of these modifications and extensions are suggested by authors themselves

from a pool of only three aircraft. Davendralingam and Crossley reduce a similar problem into a semi-definite programming (SDP) form to address uncertainty. [14] A low fidelity aircraft sizing code (Flight Optimization System , FLOPS) is used to design aircraft that operate on an 8 city network at the SoS level while maximizing airline revenue (an SoS level metric). The methodology is innovative and can handle uncertainty in constraints (refer to the Bertsimas-sim application to robust portfolio optimization [15]). Nevertheless, it is only applicable to problems with a top level objective function. We aim to design a framework that goes one step further: incorporate high fidelity simulation-based-optimization that bridges the gap between the theory, and practical application of SoS (which may or may not have a top level function to optimize). Note that even today, SoS optimization studies are not limited to conceptual ones. Azarnoush *et al*. solve a practical SoS problem involving mobile robots trying to detect a threat in an unknown environment. The problem consists of two levels – A master robot at the SoS level and a swarm of robots in the system level. The trajectory of each robot is optimized with respect to a local objective. The final SoS objective is simply a weighted sum of all the individual lower level objectives. A similar practical SoS optimization demonstration using swarm robotics is discussed in [16].

Wolf examines the simultaneous design of ships that belong to a SoS (Sea-base concept). [9] A single, top level SoS performance objective (time to transport/deliver troops or cost) drives the design. Collaborative optimization (CO) is used to decompose this problem into individual ship optimization problems. It is difficult to perform simultaneous analysis and design of each of these individual ships, which have their own (multiple) design goals and constraints, while also contributing to the overall top-level SoS performance. The author rightly identifies that designing systems independently (or ignoring SoS) may result in underutilized (excess) equipment and capability. [9] Wolf extends collaborative optimization to a case with multiple objectives, but still identifies disadvantages such as computational ineffectiveness and sub-optimal solutions with redundant capabilities due to the use of Genetic Algorithm (GA). Here again, the author was able to apply a MDO approach (CO) to this SoS optimization problem since the number of distinct levels was two. It is important for us to acknowledge here that the original implementations of the

MDO methodologies discussed here (like CO, MDF, BLISS etc.) are not limited by the number of sub-spaces or levels. The preceding discussion only introduces some specific implementations of the same. However, all these implementations are well suited to disciplinary decomposition, and may solve problems that need to be decomposed by discipline **and** level of the SoS (for example: airfoil $<$ wing $<$ aircraft $<$ swarm) with suitable modifications.

Kim and Hidalgo's research is one of a kind in that a clear approach to SoS optimization by linking multi-stage programming and MDO is offered. [13] Like in [12], the author looks at the problem as a combination of resource allocation and product design. However, only four aircraft are introduced in total during four stages (i.e. one aircraft per stage), which are themselves results of a lower level optimization problem involving design variables T/W, W/S, AR and number of passengers. Although this work also explores a bi-level decomposition of such a problem, it offers a method to extend the formulation to multiple levels (by considering them as several bi-level problems and solving the lowest level first). Nested MDO strategies (like Analytical Target Cascading (ATC) with CO used by Allison *et al*. [17]) are closest in structure to type of problems we are interested in - disciplinary decomposition through MDO to design a sub-system, followed by requirements-based assembly of sub-systems to form systems at higher levels.

We observe the following in all the aforementioned research work:

1. MDO methods are not designed to effectively captured the multi-stage aspect of SoS design beyond two levels. Usually, the levels considered are the SoS or fleet level, and the aircraft level. We recognize that the implementations of MDO approaches discussed here are problem-specific and are therefore not expected to solve generic SoS optimization problems without modifications. MDO approaches use disciplinary decomposition to design and optimize systems, and may even be modified to use other sub-spaces as a proxy for individual disciplines [2]. We intend to describe a framework that naturally describes practical problems that are multi-stage, multi-level, **and** multi-disciplinary in nature.

[2]Such modifications may give rise to multi-level or hierarchical decompositions

2. The aircraft level is generally simplified to reduce 'impractical' computational cost. It must be recognized that each level in an SoS framework is important, although finally the top-level capability must be achieved. Also, current parallel computing architectures are making computationally expensive tools faster to run.

3. As discussed by Dennis and Arroyo, [18] for some SoS problems, there may be no discernible top level objective other than to find a feasible strategy. Oftentimes, a mission-suitable SoS or a SoS with a desired capability is designed. When translated to the actual solution strategy used, this means that a candidate SoS may need to be checked or validated against this desired capability, rather than optimized with respect to some objective function. Authors use a top-level function (like cost or time/performance) so as to use existing frameworks and algorithms, or for a concrete mathematical formulation. This often takes the form of minimizing the sum of constraint violations or constraint programming. While having the option of using constraint programming for certain sub-problems in the framework adds flexibility to the approach, cost is frequently used as a top level function in SoS problems. If two options provide the same capability, the option with lower cost is desired.

4. SoS evolve with time, and therefore the design space also evolves with time. Authors have chosen to use static (or minimally changing) design spaces due to computational difficulties, and scarcity of algorithms to handle evolving design spaces.

In summary, we identify that the direct application of current state-of-the-art methodologies are not appropriate in terms of at least three essential areas, and hence may not be useful in our approach (see table 1.1). Our approach aims to push the envelope of SoS-based optimization by formally defining the problem structure, and by proposing one possible approach to solve problems that fit into this new structure.

Cramer and Frank discuss SoS analogs of popular MDO methodologies. [18] Although actual applications of these SoS optimizing frameworks were not provided, it is useful to reflect upon these comparisons (see table 1.2). This provides us with a good starting point to progress towards the final goal. Attempting to optimize SoS as a whole by coupling all

Author	Method	Features	Levels
Wolf *et al.* [9]	CO	1,2,3,4,5	2
Taylor & Weck [11]	Decomposition	1,2,3,4,5	2
Mane *et al.* [10]	MINLP, MDO	1,3,4,5	2
Nusawardhana & Crossley [12]	DP, NLP, ILP	1,3,4,5	2
Davendralingam & Crossley [14]	SDP	1,3,5	2+
Kim & Hindalgo [13]	MSP, MDO	1,2,3,4,5	2+
Marwaha & Kokkolaras [19]	MINLP, Response surfaces	1, 3, 4, 5	2
Sobieszczanski-Sobieski [20]	TLISS	1, 2, 4, 5,	3
Kim *et al.* [21]	ATC	1, 2, 4, 5	2+

Legend	
Features	**1** - Static Design Space, **2** - Deterministic, **3** - Simplified Aircraft, **4** - Required Top Level objective, **5** - Only one level of abstraction
Method	**CO** - Collaborative Optimization, **MINLP** - Mixed Integer Nonlinear Programming, **DP** - Dynamic Programming, **LIP** - Integer Linear Programming, **SDP** - Semi-Definite Programming, **MSP** - Multi-stage Programming, **TLISS** - Tri-Level Integrated System Synthesis, **ATC** - Analytical Target Cascading
Levels	Number of SoS levels considered. Here **2+** indicates that the work demonstrates the use of a particular framework / method on two levels, however the authors state that the work 'can be extended' to higher number of levels.

Table 1.1: Existing methodologies for solving similar Aerial SoS problems organized according to the authors and features of their methods.

its levels to form one massive optimization problem is impractical, although it might be achievable with an embarrassingly parallel, distributed computational set-up. This however, is not the focus of the current work. Our goal is to present an efficient framework for analyzing Aerial-SoS applications that are challenging and difficult to solve, for the purpose of improving current aerial mission scenarios or evaluating hypothetical ones. The framework we envision must:

1. handle two types of interactions – intra-level interactions (within one particular level), and inter-level interactions (across levels)

2. handle varying fidelities of sub-system and system analysis code appropriately so as to obtain high quality products (sub-systems, aircraft and SoS). Each fidelity may require inputs of varying levels of abstraction (for example, the same aircraft may have a simplified representation for use as an input to a lower fidelity aerodynamic analysis code)

3. manage the complexity of the process (computational) and the products formed (sub-systems, aircraft and SoS networks), especially when operating under a computational/resource budget.

4. recognize that the top-level capability may be an objective or just a feasibility criterion.

5. be demonstrated for three essential levels, but must also be easily extensible to 4 or more levels of the SoS.

System of Systems (SoS)	Multidisciplinary Design Optimization (MDO)
Inactive Central Authority (ICA)	Multidisciplinary Analysis
Guiding Central Authority (GCA)	Multi-discipline Feasible (MDF)
Mediating Central Authority (MCA)	Individual Discipline Feasible (IDF)
Omnipotent Central Authority (OCA)	All At Once (AAO)

Table 1.2: Equivalent SoS methods when compared to existing Multidisciplinary Optimization (MDO) methods

1.2 Targeted Aerial SoS Challenge Problems

The Northrop Grumman Global Hawk and the Aurora Flight Sciences Theseus have similar endurance values (24 - 36 hours) and similar service ceilings (65000 to 82000 feet), but have completely different airframes, due to independent design processes, parameters and payload requirements resulting in different optimal designs. [22] Often times, these *optimal designs* neglect the interactions between sub-systems (and consequently systems themselves), resulting in significantly unexplored design choices. Our view is that the hierarchical capability and function flow structure must be considered explicitly, from individual sub-systems to the systems-of-systems abstraction. The hierarchy spans operational layers of abstraction (γ-level to α-levels) and multiple levels of fidelity in each layer (represented by available tools/models in segmented boxes for each layer), illustrating the highly complex, computationally intensive and interconnected nature of designing an aeronautics SoS capability. While research efforts have focused on addressing the computational issues of integrating designs across computationally intensive sub-system level artifacts (e.g. CFD and structural analysis of a wing), there is largely unexplored area of dealing with integration and complexity management across the SoS hierarchy, in achieving an overarching capability.

Three primary integrative aspects are depicted by figure 1.2 : 1) A multi-level approach for simultaneous analysis and optimization of components at each level of the hierarchy, 2)

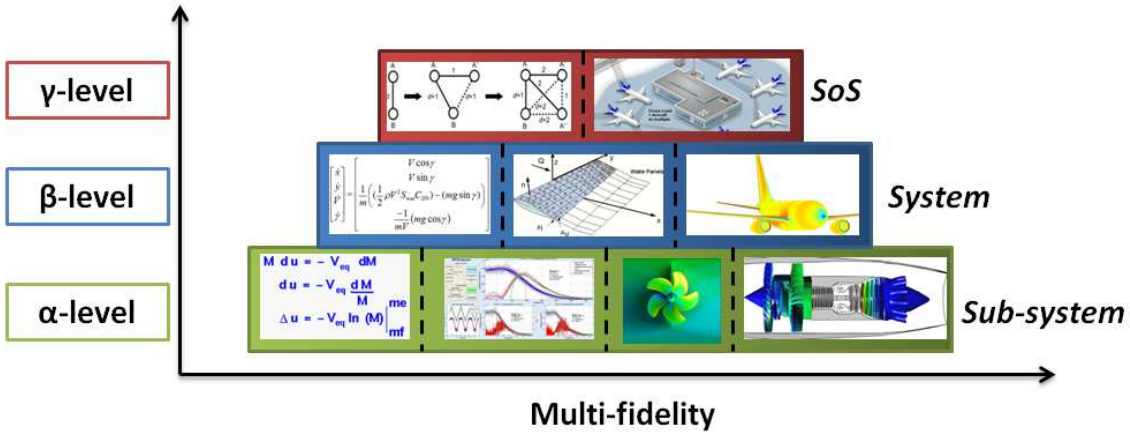


Figure 1.2.: Collection of aircraft at the SoS level (γ), Individual aircraft at system level (β) and components at sub-system level (α)

Multi-disciplinary analysis techniques to help solve the highly coupled sub-problems such as efficient propulsion-structure integration, and 3) Multi-fidelity analysis and optimization will allow intelligent use of computational resources to obtain high quality results for each discipline (structure, aerodynamics, control), at each level (sub-system, system and SoS level) of the analysis. These dimensions allow for effective management of design choices across the hierarchical levels, judicious computational resource allocation in managing information across varied levels of fidelity, and optimal selection of *baskets* of systems that best meet the SoS level capability objectives.

As a demonstration of our proposed approach, we will solve two multi-level problem that each span the three levels of physical and analytical abstraction. Although the SoS optimization problems are comprised of three levels, extension of the proposed framework to four or more levels is trivial and involves no modification of the framework or the mathematical formulation.

1.2.1 UAV-swarm for Agricultural Surveillance

UAVs are frequently assigned to dangerous, dull or dirty missions; further, their use often occurs as part of a heterogeneous fleet with multiple operators, constituting a SoS. Examples of these applications include wildfire detection and tracking, search and rescue, agricultural aerial surveillance and domestic policing. However, these commercial applications are rarely optimized in a rigorous manner so that each contributor works cohesively to achieve an overarching capability, or to properly select or develop the individual aircraft. Our SoS-centric perspective enables effectively designing heterogeneous swarms of UAVs with diverse, but complementary system level capability that cohesively give rise to a potentially greater performance at the SoS level.

Our first challenge problem is the multi-level design of a fleet of surveillance UAVs in the agricultural domain. Precision Agriculture using unmanned aerial systems (UAS) is a relatively new concept used for collecting high-resolution multi-spectral imagery for automated crop-stress detection and identification. The concept involves deploying a set of 20-40 UAVs from a mobile van, having the UAV fly over specified regions collecting imagery along the way, and returning to the van at the end of the mission. Large and complex interactions between camera capabilities, electrical power systems, UAV performance and image quality makes this a very challenging design problem. Given an area to be surveyed, a possible design problem is as follows:

1. At the SoS level, the problem is to determine the number and mix of UAVs that fly on pre-determined trajectories. These trajectories (which are part of the decision space) may depend on on-board camera characteristics, imaging requirements in terms of resolution, sun angle, and revisit time between adjacent images.

2. At the system level, the problem is to design each UAV's wing and fuselage Outer Mold Line (OML) such that the needed payload and range can be obtained.

3. At the sub-system level, the problem is to design airfoil sections along the wing root to wing tip.

This application problem is important to the field of precision agriculture and monitoring and paves the way for affordable, current and accurate Geo-information. [23–25] A majority (80%) of non-military (commercial) applications of UAVs will be for agricultural purposes (surveillance, precision agriculture, crop dusting etc.). [26] Important aspects of the problem structure and the proposed solution strategy are highlighted in Application Problem 1. Application Problem 2 presents a detailed analysis of a more realistic problem.

1.2.2 Passenger Aircraft with Minimal Noise Impact

Any increase in airport capacity must be sustainable, be safe, and abide by community noise restrictions. Recent research has revealed that aviation noise not only causes irritation to the members of a community that is close to an airport, but that continuous exposure to noise may have other long term health problems. [27] Apart from mitigating noise on the ground via insulation and the use of sound barriers, two clear solution paths may be followed:

1. Minimize aircraft (or System level) noise. This may further include:
   (a) Engine noise
   (b) Airframe noise
   (c) Noise due to flaps, slats, extensions, speed brakes and landing gear
2. Minimize operational (or SoS level) noise through improved approach procedures that take into account noise sensitive areas and mandated constraints.

Given an airport that is selected for improved approach procedures, a possible design problem is as follows:

1. At the SoS level, the problem is to dynamically determine the spacing between aircraft for minimum area of impact due to noise, such that safety is not compromised.

Thus, given noise noise optimal aircraft [3], the task is to find improved procedures can be flown at the SoS level to minimize the area covered by a specified noise contour.

2. At the system level, the problem is to minimize the airframe noise emitted by aircraft during the final approach phase with respect to level-specific decision variables such as aircraft wing, fuselage and tail section parameters.

3. At the sub-system level, we consider the aero-structural design of airfoil sections from the wing root to wing tip, as well as turbojet engine feasibility analysis.

To summarize, this application problem derives an operational approach procedure that these "noise optimal" aircraft must follow for safe and efficient flight, the aircraft themselves being assembled from a set of relevant components and sub-systems.

1.3 Research Tasks

We recognize that problems such as the ones described above in Sections 1.2.1 and 1.2.2 are important to the industry as well as academia. Take for example, the recent venture of the online shopping giant, Amazon, which is embarking on an ambitious project of creating a *delivery fleet* for delivering products to customers "within minutes". [28] Zookal Inc. plans to have a similar fleet of UAVs that will deliver textbooks in Australia next year. [29] Currently, tests are being conducted with single drones for judging the payload limit. However, with a framework for simulating the entire mission involving a fleet of UAVs or aircraft, with intricate sub-system and system design, will speed-up the development time, and reduce the amount of flight testing required. This is done by introducing improved fidelity models that can properly predict performance, thereby reducing the need for iterative improvement based on experimental test results. Additionally, a framework that is built on the strong foundations of multi-level optimization and hierarchical complexity will be efficient in highlighting potential issues and benefits at the fleet(SoS) level.

The framework we envision can be summarized as follows :

[3] In our application problem, we use far field noise intensity of aircraft in the clean wing configuration. More details can be found in Chapter 6

Multi-level Aerial SoS (ASoS) optimization testbed (Algorithms, Problem formulation, Solution strategy and Demonstrations) for critical analysis of new and existing, collaborative aircraft mission scenarios.

Although MDO-based methods can be tweaked for solving certain classes of SoS optimization problems[4], we base our framework on a unique **mathematical formulation** that fits this problem type, and develop relevant **algorithms** to solve discipline-specific or level-specific sub-problems. We have identified the shortcomings of existing methods for handling such large scale Aerial SoS problems (see section 1.1), and present a framework that involves the following tasks.

1. Describe typical steps involved in setting up and solving SoS optimization problems. Also set up and demonstrate use of overarching *hierarchical complexity metric* to make the process and product tractable (**Chapter 2**).
    - The *process* refers to intelligent management of computations, data transfers, convergence criteria and auto-adjusting parameters that control the effectiveness of the algorithm.
    - *Products* are components in each level of the SoS having multiple levels of abstraction per level of representation.
2. Develop a *core optimization algorithm* that addresses computational difficulty and the need for global search (**Chapter 3**)
    (a) Each level involves the use of expensive, discipline-specific black-box tools that are usually avoided, or replaced by lower fidelity tools for conceptual search.

[4]See chapter 2 for discussion on the difference in problem structure between MDO and SoS optimization problems

(b) The SoS level also needs an efficient optimization algorithm for identifying feasible networks (of discrete choices aircraft) for that particular mission.

A custom algorithm that can be used in both of the above scenarios is needed.

3. Explore and present the concepts related to *Evolving Design spaces* in the context of optimization from the perspective of application and theory. (**Chapter 5**)

4. Integrate and demonstrate the proposed testbed for a *Precision-Agriculture UAV swarm* and a *Noise efficient aircraft and arrival procedures*. (**Chapters 4 & 6** respectively)

1.3.1 Areas of Contribution

The three main areas of contribution of this research work are summarized below:

Template for other SoS optimization problems

This research seeks to achieve a SoS based design paradigm, via the simultaneous integration of three essential levels of hierarchical optimization. The paradigm serves as a generic template for SoS problem formulations for the purposes of optimization. We define the structure of an SOS optimization problem, and differentiate it from MDO and Nested-MDO problems. Given this new problem structure, a novel solution method is introduced (see Chapter 2 for details). It merges tenets of intelligent optimization techniques and hierarchical, interrelated levels to allow for the design and development of an SoS. The template also allows for the assessment of behaviors under conditions of uncertainty. Our research demonstrates this through solution of the aforementioned challenge problems. The challenge problems can be generalized and extended, and a standardized mathematical formulation for a multi-level, multi-fidelity problem is also presented.

Algorithm Development

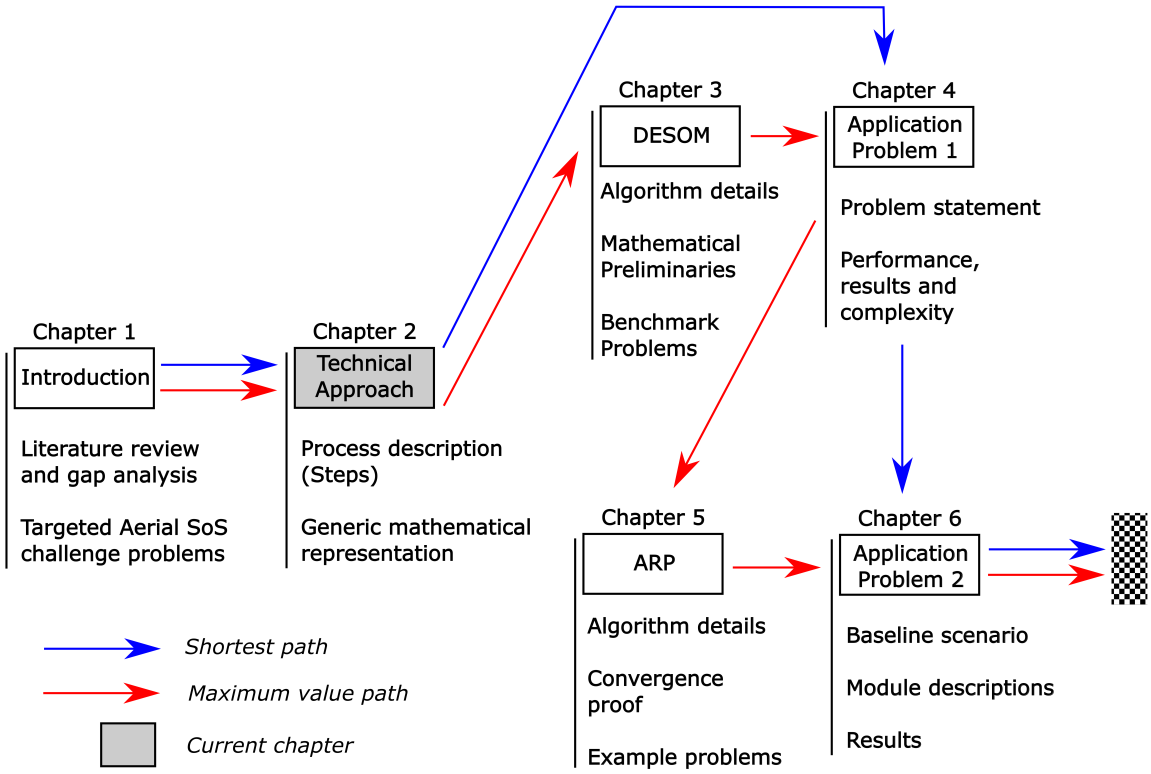
This research will result in algorithmic advancements that address the complexities in dealing with multi-fidelity tools across the hierarchical spectrum of designing an SoS entity. Our optimization algorithm, Differential Evolution with Self Organizing Maps (DE-SOM) will form the backbone of the entire methodology. The sub-problems in this research such as shape optimization of entire aircraft, detailed engine analysis etc. will require the creation of new optimization and analysis tools.

Development of an optimization algorithm at the SoS level is a challenging task since the concept of SoS itself is fluid in nature, and involves qualitative aspects such as policy, group judgment and concurrent operations. Sobieszczansky-Sobieski acknowledges that potentially incompatible design objectives may occur at every level of a decomposed SoS, and introduces a tri-level algorithm for hierarchical optimization. [30] Other options for multilevel optimization include Collaborative Optimization (hierarchical decomposition along disciplinary boundaries), and Concurrent Sub-Space Optimization (temporary decoupling of subsystems). [31, 32] These methods are promising solution strategies that may need to be suitably modified for application to certain classes of SoS optimization problems (to be explained in detail in Chapter 2). [33–35] However, we propose a solution strategy that is generic, and may be used to solve all SoS problems that are posed as shown in our mathematical stencil. Generic SoS problem formulations developed will be leveraged in the development of solution strategies and algorithms for generating holistic solutions valid across all levels of the SoS.

The vastly unexplored area of evolving design spaces becomes relevant in the context of SoS optimization. Two methods of improving designs by extending a design space are explored - 1) Numerical continuation and 2) Random projections. Each of the challenge problems adopt one of these strategies to implement optimization in an evolving design space. Also, in the course of solving application problem 2, a new hybrid method for calculating optimal trajectories is developed. The aircraft are then *flown* in a custom SoS CFD environment.

Computational Resource Monitoring

The problems we attempt to solve are computationally intractable in nature, due to the presence of multiple simulation tools, use of evolutionary algorithms and the sheer number of sub-processes needed in order to complete a top-level process. Also, two computationally similar problems may suit completely different speed-up methods - for example, certain problems suit parallelization using CPU cluster, whereas others may suit GPU-style computations. To showcase this variety in structure, we carefully choose suitable computational architectures available freely or commercially (Eg. in MATLAB). In a resource constrained scenario, a simple, yet novel hierarchical complexity metric adapted from sources in literature is used to monitor these processes and allocate available resources intelligently (refer to chapter 2 and the application problem 1 in chapter 4). On the other hand, a sequential or iterative computational set-up may suit certain other problems (refer application problem 2 in chapter 6). In addition, our framework supports the use of multi-fidelity tools through the use of a Value-of-Information (VoI) based decision logic that connects the time required for an analysis and the value of utilizing that level of fidelity for analyzing a design.



In this chapter we introduce the mathematical framework that will be used to represent SoS optimization problems, and also discuss some key steps, features and foundations of the approach including Platform-based Design, Hierarchical Complexity, multi-fidelity analysis and the use of Value of Information. Specific optimization-related algorithms developed will be discussed separately in forthcoming chapters.

2. Technical Approach

This chapter introduces concepts related to SoS optimization such as the problem structure (Section 2.1) and its relation to MDO and nested-MDO problems. Features specific to SoS optimization problems that differentiate them from multi-level, multi-disciplinary problems are also described. Finally, we describe a generic solution strategy that can be used to solve problems that fit into the generic mathematical template presented (Section 2.2).

Special features are added to a standard multi-level optimization problem to represent an SoS optimization problem. Since SoS optimization problems are expected to be expensive by nature, a hierarchical complexity metric is proposed to control and manage the process during actual implementation. Foundational elements of this framework such as Platform-based Design (PBD) paradigm, evolving design spaces, multi-fidelity analysis and hierarchical complexity are also described as part of the process description.

2.1 SoS Optimization : Problem Structure

Figure 2.1 shows an Euler diagram for the set of optimization problems. [1] This diagram **does not** differentiate or classify these problem types based on problem features. It simply allows us to defines sets of problems that we are interested in based on the mathematical structure, and then show valid subsets and intersections. MDO problems (which are optimization problems themselves) are examples of a larger set of problems - Nested MDO. A network or a hierarchy of MDO problems is a nested MDO problem. In this sense, MDO problems are special instances of Nested MDO problems, similar to a node being a special instance of a network (a network may have a single node too, therefore, all MDO problems are special cases of nested MDO problems, with the number of MDO's

[1] This is similar to the popular P vs NP Euler diagram. Note that this diagram is only for demonstration purposes and does not come with any formal proof

involved = 1). These Nested MDO problems share its general problem structure with SoS optimization problems, that we are interested in here. However, unique additions to the structure of SoS optimization problems differentiate them from Nested MDO. The intersection between Nested MDO and SoS optimization problems represent the set of problems that may be solved by existing techniques. Therefore, Nested MDO problems may manifest themselves as SoS optimization problems. This is true when the special features of an SoS problem structure is non existent, but the problem still involves more than one interacting MDO problem. We are interested in the set of problems that can be represented by SoS optimization problem structures alone. Problems in the intersection of SoS and Nested MDO may very well be solved by existing methodologies. We further explain the difference between these problem types in the following paragraphs. Note that more comprehensive or general problem formulations may exist, but the discussion below uses a minimal representation to demonstrate the difference in problem structure.

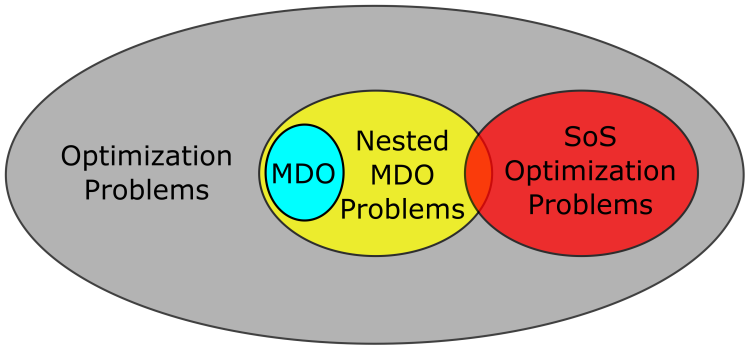


Figure 2.1.: The set of SoS optimization problems and their relation to MDO and nested-MDO problems

Let us first recognize that the set of SoS optimization problems have a unique structure; this differentiates this problem set from other related problems such as MDO problems, nested-MDO problems and optimization problems of the general variety. A general optimization problem may be formulated as:

$$\begin{aligned} x^* &= \arg\min_x f(x) \\ S.To \;\; g(x) &\leq 0 \\ h(x) &= 0 \end{aligned} \tag{2.1}$$

where x^* is an optimum point with respect to an objective function f, subject to equality (h) and inequality (g) constraints.

An MDO problem $\mathscr{P}$ uses global as well as discipline specific variables, constraints and copies of variables to reach a solution and may be written as (from Martins [176]):

$$\begin{aligned} \{x_0^*,\, x_i^*\} &= \arg\min_{x,\hat{y},y,\bar{y}} f_0(x,y) + \sum_{i=1}^{N} f_i(x_0,x_i,y_i) \\ S.To \;\; g_0(x,y) &\leq 0 \\ h_0(x,y) &= 0 \\ g_i(x_0,x_i,y_i) &\leq 0 \\ h_i(x_0,x_i,y_i) &= 0 \\ \hat{y} - y_i &= 0 \\ \mathscr{R}_i(x_0,x_i,y_i,\bar{y}_i,\hat{y}_{j\neq i}) &= 0 \end{aligned} \tag{2.2}$$

where $()_0$ pertain to global constraints and variables, and $()_i$ pertain to discipline or subspace specific ones. x and y are design variables and coupling variables respectively. Copies of the coupling variables ($\hat{y}$) are enforced via consistency constraints $\hat{y} - y_i = 0$. State variables $\bar{y}$ may occur in physics-based solvers relevant to *that* discipline (eg: Navier-stokes CFD solver in the Aerodynamics discipline that may need to "converge" below some tolerance level). **The result of an MDO problem (problem $\mathscr{P}$) is an optimized product, which may then be used as part of an assembly.** [2] Here, the local and global design variables describe the product (x_0^*, x_i^*). It is important to note that a product as a result of this MDO problem may be interchangeably denoted as $\{x_0^*,\, x_i^*\}$ or $\mathscr{P}$ itself for brevity in the discussion that follows. For example, $\mathscr{P}$ may represent an aircraft.

[2] Other authors may refer to a "product" as an "object", a "component" or a "sub-assembly"

Next, we consider a *Nested*-MDO problem (defined as interacting MDO, or hierarchical MDO, or a network of MDO problems) using the above notation as :

$$\begin{aligned} \mathscr{P}_0^* &= \arg\min_{\{\mathscr{P}_j\}} F(\mathscr{P}_0) \\ S.To \;\; \mathscr{P}_0 &= \{\mathscr{P}_j,\ \forall j = 1:M\} \\ A(\{\mathscr{P}_j, \mathscr{P}_k\}) &= 0,\ \forall j,k = 1:M \\ C(\{\mathscr{P}_j, \mathscr{P}_k\}) &= 0,\ \forall j,k = 1:M \\ V(\{\mathscr{P}_j\}) &= 0,\ \forall j = 1:M \end{aligned} \tag{2.3}$$

where $\mathscr{P}_0$ is the outermost problem, $\{\mathscr{P}_j\}$ are interior problems which may themselves be MDO problems, F is a measure of performance of the assembly $\mathscr{P}_0$, A are interface constraints or physical connections, C are communication constraints, and V are analysis or validation constraints. The constraints A and C describe how two solutions $\mathscr{P}_j$ and $\mathscr{P}_k$ communicate or connect, whereas validation functions V may be required to test interior solutions prior to their use in the outermost problem $\mathscr{P}_0$.

SoS optimization problems are similar in structure to Nested-MDO problems, but have several unique features shown below:

$$\begin{aligned} \mathscr{P}_{0,l}^{t,*} &= \arg\min_{\{\mathscr{P}_{j,l}^t\}} F(\mathscr{P}_{0,l}^t) \\ S.To \;\; \mathscr{P}_{0,l}^t &= \{\mathscr{P}_j,\ \forall j = 1:M+\Delta M\} \\ A(\{\mathscr{P}_{j,l}^t, \mathscr{P}_{k,l}^t\}) &= 0,\ \forall j,k = 1:M+\Delta M, \forall l \text{ at time } t \\ C(\{\mathscr{P}_{j,l}^t, \mathscr{P}_{k,l}^t) &= 0,\ \forall j,k = 1:M+\Delta M, \forall l \text{ at time } t \\ V(\{\mathscr{P}_{j,l}^t\}) &= 0,\ \forall j = 1:M+\Delta M, \forall l \text{ at time } t \\ \mathscr{P}_{j,l}^{t+\Delta t} &= \mathscr{P}_{\text{expert}} \end{aligned} \tag{2.4}$$

where experts may introduce new products $\mathscr{P}_{\text{expert}}$ at a particular time t (which may be treated as solutions of an MDO problem), solutions may now exist at multiple levels of fidelity or "abstraction", and new problems may be introduced through an evolving design

space (increasing number of choices to pick from, or changing the optimization problems that result in the product itself). The aforementioned features allow us to use the same structure as nested MDO problems while also better reflecting the nature of problems found in practice.

It is easy to see that these additional, unique features arise because of SoS characteristics, for example :

1. Operational and Managerial Independence - Each level designs and optimizes products (or solutions to MDO problems) independently. However, an interaction exists through instructions that flow across and within levels (to be detailed shortly).
2. Evolving Nature - introduction of optimization in an evolving design space
3. Levels of abstraction - a product may have different interpretations at different levels
4. Stakeholder involvement through designs that are provided by experts
5. Multi-stage aspect due to the added dependence on time t

Next, we provide details about the steps involved in our approach for the solution of SoS optimization problems.

2.2 SoS Optimization : Solution Strategy

We motivate our approach and associated process description with the precision agriculture UAV challenge problem introduced in section 1.2.1. Solving an SoS optimization problem will generally involve the following steps:

1. Capability Flowdown
2. Multi-level, multi-disciplinary Analysis Management
   (a) Forming sub-system sets
   (b) Populating the system library

(c) SoS-level synthesis

3. Multi-Fidelity analysis

4. Complexity Management

Note that depending on the problem type, the phase of design and the quality of models used in the process, some of the above steps may not be used. For example, a problem that does not operate on a computational or time budget may not require the management of process complexity (although product complexity may feature in one of the problems) . On the other hand, a problem that does not involve multi-fidelity tools may not require our method for multi-fidelity analysis management.

2.2.1 Step 1 - Capability Flowdown

The capability of our UAV-based precision agriculture challenge problem may be decomposed into two basic SoS level capabilities (surveillance and data processing) and further decomposed to system and component requirements (see Figure 2.2). Apart from the capabilities explicitly derived from the SoS level, there may be capabilities that are application- or level-specific. For instance, growers may require precision maps containing information about nutrition, water, insects or fertilizers. Information about insects can be obtained with a fleet capable of providing higher effective resolution, whereas a coarser resolution map is sufficient for obtaining information relevant to optimizing water distribution and irrigation channels. These implicit and explicit capabilities at the system level will correspond to several items in the material and sub-system library. The capability flowdown map generated is a cursory representation of the actual computational framework.

The capability flowdown step translates the required capability from the SoS level, to specific requirements that the lower levels must satisfy. Lower levels may have existing systems that satisfy this requirement. Otherwise, new systems are designed or optimized by triggering or activating lower levels to satisfy these requirements. Of course, these requirements are modeled as performance thresholds, or constraint violations. This allows

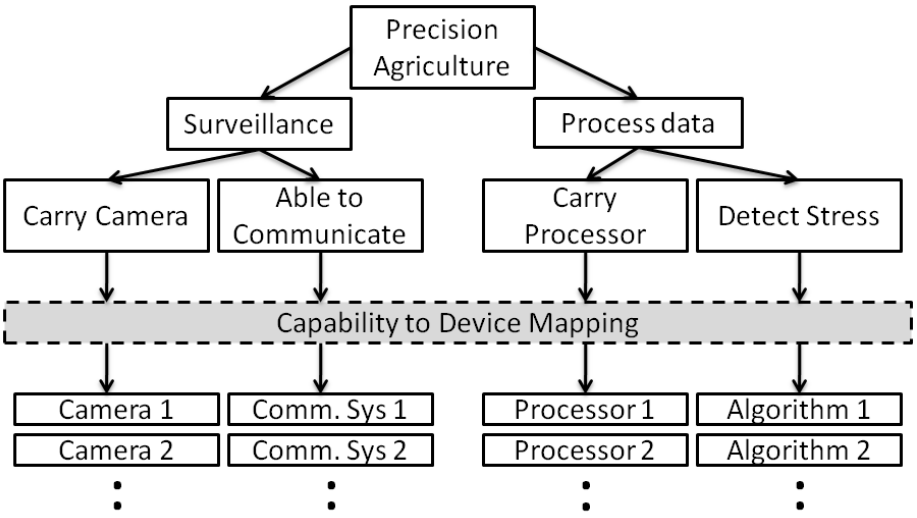


Figure 2.2.: Sample capability flowdown for the precision agriculture SoS example

for under- or over-achievement also. Capability flowdown is therefore matched by a flow in the upward direction by products in the sub-system, system or SoS level.

Here, we clarify the difference between 'levels of an SoS' and 'levels of abstraction' at a level in the SoS. Consider the system level (with a library or collection of assembled aircraft). The aircraft at this level may be represented in several ways (several fidelities of representation):

- By the span and aspect ratio of its wing, and fuselage length.
- By a detailed CAD model representing the aircraft geometry to a high level of detail.

Each of these representations (*abstractions*) are useful for tools to be used at *that* particular level of fidelity. For example, a very detailed, high resolution geometry is required for a CFD tool, but the a lifting-line theory based code analyzing the same aircraft may not use this level of detail. *Levels of the SoS*, on the other hand, correspond to the level that a product (for example, an airfoil or an aircraft) belongs to. This brings us to one the foundations of this research, Platform-based design.

Platform Based Design

A platform is defined as a library of components, across multiple hierarchical levels, that can be assembled to generate a design at that level of abstraction. [36] Each element in the library is associated with some performance parameters, supported functionalities and interfaces. The library as a whole is a parameterization of the design space. The PBD design process is neither top-down, nor bottom-up; rather, it is a meet-in-the-middle process. [36] The PBD methodology was successfully adopted by several automotive manufacturers in Europe for managing development time and increased complexity, but the methodology used is generic, and not limited to automotive applications. [37] Two key principles of PBD make it useful in the design of complex aerospace systems:

1. It addresses complexity by introducing layers of abstraction, and
2. It separates out the specification of functionality and architecture at each layer.

We expect the updates at the sub-system, system and SoS levels happen independently in our design process. The architecture choices used may depend on the style of central authority (omnipotent, in the case of Application problem 1, and mediating in Application problem 2).

PBD is also linked to the use of libraries of components that exist in each level in our framework. This collection or library of components exist at multiple levels of abstraction. For example, the system level may append a new aircraft to the existing collection or library of aircraft that it offers as a choice to the SoS architect. However, the aircraft analysis code that exists in the system level may "see" this aircraft at a much higher fidelity level or resolution, than the SoS level, where important, discernible characteristics of the aircraft may be sufficient for use in SoS level performance evaluation codes.

2.2.2 Step 2 - Multi-level, Multi-disciplinary Analysis Management

The establishment of a capability flow down in Step 1 generates requirements data that flow to each hierarchical level. To support our design paradigm, analysis and simulation

tools should allow designers to combine models from different domains into integrated system and SoS level models, and allow models of components and sub-systems (including algorithms) to evolve throughout the design process from conceptual design to detailed design. This requires the development of a multitude of system models at SoS levels, as well as models involving combinations of disciplinary analyses and optimization blocks.

Step 2a. Forming Sub-System Sets

Once all relevant sub-systems and material types are selected from the corresponding libraries, they are modified and tuned for system integration - both across the hardware and software spectra of the sub-system hierarchies where components in the library must have an interface specification that describe compatibility requirements with other components. Optimization at the sub-system level introduces a range of operational modes of each sub-system. An engine (sub-system) designed to provide 1000 lbs of thrust can be improved to provide 1100 lbs ($1.1x$) at a lower rate of fuel consumption. Engines corresponding to optimal ($1.1x$) and sub-optimal ($1x$ to $1.1x$) configurations form a flexible set of designs. Flexible sub-systems help form flexible systems, and this flexibility is implicitly transferred up to the SoS level. Although this flexibility may or may not be explicitly desired, higher levels may test the hypothesis that *sub-optimal sub-systems may be assembled to form optimal systems* in an SoS optimization framework. Thus, at each level, we have sets of co-related designs instead of disparate design points. Our use of evolutionary optimization algorithms allows for the grouping of sets of feasible designs (population members) that have varying performance characteristics.

Step 2b. Populating the System Library

Unlike the sub-system and material libraries, the system library that is to comprise of a set of feasible aircraft, can initially be empty. The library is dynamically updated as and when feasible, mission certified aircraft are available via the simulation-based design optimization. The creation of primary designs will involve global search using multi-fidelity

and multi-disciplinary tools. An aircraft is assembled using products delivered by analysis blocks the sub-system level, and only feasible or optimal aircraft are added to the library of aircraft for selection by the SoS level.

A core capability required for composing the system models is an understanding of system behavior under nominal and off-nominal conditions. We propose to simulate the behavior of the system under nominal and off-nominal operating conditions to reason about component failures and functional losses, and assess their impact downstream at the sub-system level as well as upstream at the SoS level. [38, 39] This is achieved through robust counterpart optimization and sensitivity analysis. Due to its central position in our Aerial SoS example problems, System level is critical to the flow of information upstream and downstream.

Step 2c. SoS-level Synthesis

The synthesis of aircraft systems, with capabilities, requirements and risk (for example, technology readiness level (TRL)), bears much resemblance to that of a portfolio problem where the objective is to select collections of systems, each subject to specific rules of connectivity and requirements, that work cohesively in providing some overarching SoS level capability. These systems or *nodes* and associated rules of connectivity or *links* can be selected judiciously using optimization paradigms that can balance potential benefits of the resulting SoS architecture against manifestations of various measures of operational risk. This work will employ recent algorithmic advances in optimization to enable navigation of the complex combinatorial trade-space, and aid in the selection of complete collections of systems that constitute an optimal SoS. [40, 41]

When a new aircraft is added to the system library, it is verified to be mission-suitable, and is then added as one of the nodes in a potential SoS network. However, as new elements are added to a system library, the number of potential SoS configurations grows (at worst, by n^2 where n is number of nodes in system library). Further, the sub-system, system and SoS level updates may happen asynchronously (depending on the computational set-

up), thus making the design space dynamic in nature. Here, the design space becomes dynamic since one of the design variables in the optimization problems considered could be the choice a component from a library (that increases in size, thereby allowing more than the original number of options / choices). Depending on the mission type and the objective function, homogeneous or heterogeneous networks may be preferred. If cost (in $) is the objective function in the present precision agricultural example, it may be useful to delegate specific functions to different sets of aircraft. Such trades are addressed via an overall complexity management process.

2.2.3 Step 3- Multi-Fidelity Analysis

Fidelity is the degree to which a model reflects the behavior of a real system. *Model A* is said to be at a higher fidelity than *Model B* if more physical phenomenon are modeled in an effort to replicate real physics of the analysis involved. Accuracy, on the other hand, applies only to the results of a simulation. It answers the question, how close are the results obtained to the real value of the parameter(s) being measured? [42] Two main bifurcations of multi-fidelity modeling used in optimization are 1) constant parameter dimension, and 2) variable parameter dimension. [43] The difference lies in the dimension of 'inputs' (x) to a lower fidelity model $\hat{f}$ and that of a higher fidelity model f. While Robinson *et al.* describes typical methods used in 2), our simulations are more suited to 1), where the same input vector may be interpreted and used at different levels of fidelity. State-of-the-art methods in this class of multi-fidelity global optimizers include Efficient Global Optimizer (EGO) and Value-based Global Optimizers (VGO). [42, 44]

Operating in a constant parameter multi-fidelity space allows us to use state-of-the-art methods that are shown to have provably convergent formulations that are equivalent to optimization in the highest fidelity level. [42, 44] Although the proofs of these algorithms are provided for convex and differentiable functions, demonstration problems have shown their effectiveness for non-convex functions as well. Both EGO and VGO use a surrogate model as a backbone to derive properties relevant to their global optimizer in a multi-

fidelity environment (DACE for EGO, and Gaussian processes for VGO). While EGO uses the Expected Improvement (EI) metric, VGO uses the more versatile Value of Information (VoI) metric. VoI allows for the use of multiple fidelity levels, whereas EI (and the global optimizers that use this) allows only one higher fidelity model. [42] Furthermore, EI is associated with an arbitrary, user-defined stopping criteria whereas VoI's stopping criteria is intuitive (that is, stop when cost $\geq$ benefit, or VoI ≤ 0). VoI suits our purpose since it is a utility function that encompasses not only the final product, but also the cost of the process needed to design that product. We modify the VoI formulation found in [42], since the original formulation suggests the next point to be sampled as well as the fidelity level, whereas our 'next point' is chosen by the evolutionary optimization algorithm DESOM2. [45] Let the function we wish to maximize be $f : \mathbb{R}^n \to \mathbb{R}$. Let the previous sample points until the current time-step be given as $\{x_k\}$, with corresponding function values $\{f_k\}$, regardless of function fidelity. The value of an analysis at fidelity i is given as: [42]

$$v_i = \max(E(f|f_k), f_{max}) - f_{max} \tag{2.5}$$

Given a cost C_i for using a model of fidelity i for evaluating $f(x_i)$, the value of incremental gain G_i may be given as:

$$\begin{aligned} G_i &= v_i/C_i \\ &= \frac{\max(E(f|f_k), f_{max}) - f_{max}}{C_i} \end{aligned} \tag{2.6}$$

We diverge from the original VoI formulation at this point since 1) we intend to use existing sampling methods or global optimizers, and 2) Our decision on fidelity level to use is based on a single model that represents outputs from sample points at all fidelity levels, and our cost is measured in units of time (thus, we avoid writing $G_i = v_i - C_i$). We use results reported in [45] (summarized in chapter 3) , where we derive the expected value of a new population member x_i generated by an evolutionary optimizer. Note that one would be aware of the cost C_i, which is measured in units of time since prior tests may be

conducted to measure the average time taken to complete an analysis at a particular fidelity level. Assume that the function f is convex and differentiable. [3] Then from [45],

$$E(x_i) = CR \cdot x_{av} + (1 - CR) \cdot x_i \tag{2.7}$$

where CR is the crossover probability, and x_{av} is the average location of the population members (where we expect to finally find the optimum $f^* \geq f_{max}$. Depending on the optimizer used, we may not find a $f^* \geq f_{max}$. Now, using Jensen's inequality for convex functions, we obtain:

$$\begin{aligned} E(f(x_i)) &\geq f(E(x_i)) \\ &= f(CR \cdot x_{av} + (1 - CR) \cdot x_i) \end{aligned} \tag{2.8}$$

Since f is a convex function, $f(CR \cdot x_{av} + (1 - CR) \cdot x_i) \leq CR \cdot f_{max} + (1 - CR) \cdot f(x_i)$, which gives:

$$E(f(x_i)) \geq CR \cdot f_{max} + (1 - CR) \cdot \hat{f}(x_i) \tag{2.9}$$

where $\hat{f}$ is the potential method surrogate model used by DESOM2. The gain G_i is then given as:

$$\begin{aligned} G_i &= \frac{\max(CR \cdot f_{max} + (1 - CR) \cdot \hat{f}(x_i), f_{max}) - f_{max}}{C_i} \\ &= \frac{\max(CR + (1 - CR) \cdot \hat{f}(x_i)/f_{max}, 1) - 1}{C_i/f_{max}} \end{aligned} \tag{2.10}$$

The incremental gain (G_i) values corresponding to each fidelity level is used during an optimization run to decide the fidelity of the model i^* to be chosen for a particular population member x_i. Let the fidelity levels be represented from lowest (1) to the highest available (N_f). The following decision logic is used in correspondence to the G_i values.

[3] Although the assumption that f is convex and differentiable is restrictive, this allows us to perform some more

$$i^* = \begin{cases} \arg\max_i \{G_i\} & \text{, if } \min(G_i) > 0 \\ \arg\max_{i \setminus i'} \{G_i\}, i' = \arg(G_i = 0) & \text{, if } \min(G_i) \geq 0 \\ 1 & \text{, Otherwise} \end{cases}$$

If model error is known, a worst case function value may be used instead of $\hat{f}(x_i)$. Results presented here are purely theoretical with example problems that indicate effectiveness outside these assumptions. Our implementation of the VoI metric and the formulae discussed here use results pertaining to the DE-SOM algorithm. The implementation using other global optimizers has not been explored here. Also, depending on the application problem, the C_i term may not have the units of time. Thus, a careful modification of these metrics is required for effective and correct usage.

Bertschinger *et al.* use VoI in their work describing information geometry on non-cooperative games. [46] The authors show how the intuition that "additional information is always valuable" may be violated in games involving multiple players or bounded rational behavior. So, a higher fidelity source has higher information content (lower noise). The authors in [46] briefly describe single player games and the calculation of VoI, which is related to our implementation of a single player choosing an appropriate fidelity level. They define marginal value or improvement (VoI) as the projection of the gradient of the expected utility (here, $\hat{f}$) and the gradient of VoI. Extensions of the work presented here will incorporate such metrics for multi-fidelity optimization.

2.2.4 Step 4- Complexity Management

The process then proceeds to the management of inherent complexities that manifest across the three sub-steps (sub-levels) of (2a-c). The pressure to conduct affordable development programs requires that the orchestration of complex systems be planned to significant detail, well in advance of actual design studies. This is because complex distributed systems can exhibit behavior that is not easily predictable. [47] The RAND corporation attributed the rise in costs of fixed-wing military aircraft to increased complexity. [48] Due

to the dynamic nature of interactions seen in tightly coupled aircraft sub-systems, it is difficult to minimize the complexity of a system at the conceptual design stage. [37] In our research framework, complexity takes on two distinct forms:

1. Product Complexities - associated with operational complexity of assemblies, or networked components in all levels of the SoS architecture, and

2. Process Complexities - associated with computational resources required on information systems (IT) that directly support the development of the end operational SoS architecture.

The challenge of developing future complex systems in a cost-effective manner can addressed by a new generation of multi-level design processes and tools. [37] A multi-level design tool may be used study the scale, impact and behavior of system interactions early in the design process to ensure that complexity is understood before making key design decisions. While the complexity associated with each of the two noted forms requires domain specific knowledge for management, they share salient features. These facets of complexity management are described next.

2.2.5 Facets of Complexity Management

Quantification of Complexity

Complexity of a system has traditionally been quantified based on information theory as the measure of information content of the system, or by the uncertainty present in it. [49–51] Summers *et al*. model complexity by the degree of interconnections between various components of the system. [50] Although it may seem convenient to model complexity at each level in this problem using a network-centric approach, this disassociates one level from another. Initially, the conversion of a generic top-level capability description to an actual abstraction level design form can be controlled by using the 'designing effort' metric defined by Braha *et al*. [52] Due to the multi-disciplinary nature of the design process, abstract (i.e, low-fidelity) complexity metrics may serve as an early indicator, while

detailed network based metrics (high-fidelity) may be used at later stages of the design process. Apart from metrics that describe the process (like designing effort) and the information content (network based), an overall complexity must be defined for the hierarchy that comprises of all three primary levels.

To manage process complexity, we adapt the model of hierarchical complexity, a framework for scoring the complexity of task execution based on how information is organized. [53] It can be made to account for evolution of systems by recognizing that their patterns are comprised of tasks, performed at specified orders. This metric will be used to control the number of times a library at a particular level is accessed, or the time allotted to a specific task such as sub-system optimization. The entire process is quantified by orders, tasks, stages, and performance. The functional decomposition of the process reveals several primary functions (such as architecture evolution and system optimization) and some secondary functions (such as a library item notification). The *order* portion of the metric quantifies the direction and frequency of information flow from one function to another. The *task* portion quantifies the difficulty of a particular function to process information and provides results to other parent or child functions. One can imagine that initially the sub-system and system levels may be more active than the SoS level since the corresponding libraries are still being populated. These libraries may saturate and the SoS level may involve more processing. This aspect is captured by the *stage* portion of the metric. Lastly, each function may process information at different rates depending on the problem being solved. For example, CFD analysis may involve meshing different geometries of aircraft while meshing speed depends on the geometry itself. Thus a *performance* portion is also required to completely capture the notion of hierarchical complexity of the proposed design process.

Complexity Distribution

The ability to quantify complexity is as important as the process of minimizing and allocating complexity ‘budgets’ to various levels. The complexity of components present

in each level needs to be distributed according to the objective function and constraints. In the aerial precision agriculture SoS example, suppose the objective is to minimize the sum of system cost and operational cost of the SoS. An SoS involving several small, cheap, and re-usable systems (SoS-1) may be more optimal than another SoS involving two high-performance surveillance aircraft (SoS-2). At the SoS level, SoS-2 is much simpler than SoS-1 since it contains just two sophisticated systems capable of even completing individual missions. However this impression is reversed when we move to lower levels of the hierarchy. The sophisticated nature of systems of SoS-2 arises from the fact that the sub-systems are more complex. Thus, complexity considerations can eliminate the need to construct and evaluate an SoS based on a time-consuming virtual simulation, if it is likely to result in an unfavorable objective function value. Monitoring performance and complexity in the vertical and horizontal dimensions is only important when both time, and the number of computational resources are limited (as in application Problem 1). Here, the available set of processors schedule jobs pertaining to each level, and all levels of the problem share the same computational resources. Thus Application problem 1 uses a parallel computing setup that allows all levels to progress in parallel. On the other hand, when either time, or computational resources are limited and distributed, a sequential parallel architecture can be used with level-specific workers that wait for data from other levels (as in application problem 2).

Before we quantify our hierarchical complexity metric, we define the following terms:

- *Task* - An analysis or a job (or some form of computation) to be completed. It holds a value of 1 if it is completed, and 0 if it is not.
- *Order* - Ideal forms of complexity or performance evolution with time prescribed by experts.
- *Stage* - Refers to an actual task performed with respect to an *order*
- *Classical Information Complexity* (C_h) - This 'horizontal' or component-specific form of complexity is the number of yes-no questions it takes to describe a prod-

uct. For our use, this is given by

$C_h =$ Components + Links.

- *Hierarchical Complexity* (C_v) - This 'vertical' complexity is defined by the number of processes required before, and in order to perform a more complex process. [53] This is given by

  $C_v = \sum_{i=sos,sys,sub} \text{Tasks}_i$

- *Product Performance* - Performance of a product (component) at a particular stage (horizontal). For example at the SoS level,

  $P_h = \overline{f_{sos}}(X_{sos})$

- *Process Performance* - Performance of the tasks to be performed at each level at a particular time instant (vertical)

  $P_v = \sum_{i,j} \frac{\text{Fidelity}_i \times \text{Task}_j}{\text{Time to complete Task}_j}$

  Note that this calculation is valid for problem set-ups where the user is aware of various fidelity levels, and is able to calculate the Value of Information (VoI). In this case, Fidelity_i may be replaced by VoI_i. For cases where the VoI metric is not suitable, other methods must be used to indicate a difference of fidelities. Note that a careful and appropriate modification of the formulae for P_v is required with the use of VoI_i as a metric for fidelity since our implementation already factors in time (as cost, C_i), in addition to the time to complete Task_j.

Our metrics for horizontal and vertical product and process performance is simple and intuitive and used to direct the next step of the SoS optimization framework such that

> *Products* / **processes** below a predicted performance threshold or above a complexity threshold are not *formed* / **scheduled**.

This is done by controlling the value of stage and order of hierarchical performance (H_p) and hierarchical complexity (H_c):

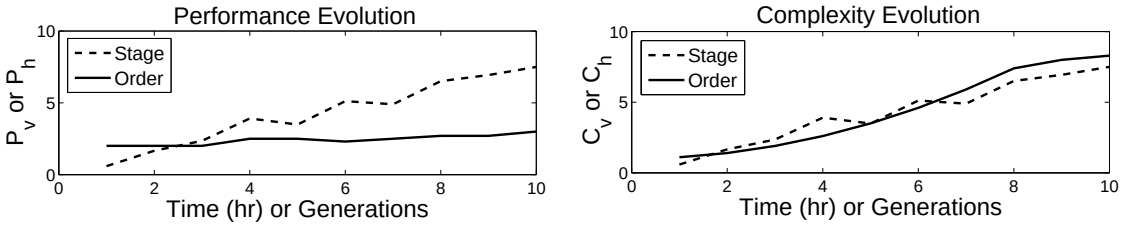


Figure 2.3.: Schematic showing evolution of stage and order of complexity and performance with respect to time or number of generations of the framework.

$$H_p = \frac{\text{stage}(P_i)}{\text{order}(P_i)}, i = v,h \text{ and } H_p \geq 1 \tag{2.11}$$

$$H_c = \frac{\text{stage}(C_i)}{\text{order}(C_i)}, i = v,h \text{ and } H_c \leq 1 \tag{2.12}$$

An example time history of the framework trying to control the values of H_p and H_v is given in figure 2.3. It is desired that the performance (vertical or horizontal) is maintained above a threshold or order (left), and that the complexity is below some corresponding order (right).

2.3 Mathematical representation of the SoS Optimization Framework

Section **??** described the structure of SoS optimization problems in general. As a reminder, the structure resembles a network, hierarchy or nested system of MDO problems that may evolve with time. However, suggesting a mathematical structure for the most general case of nested MDO problems is not the focus of this work. First, we must recognize that SoS are hierarchical in nature. Thus, a tree or hierarchy of problems is most suitable for building a mathematical stencil for SoS optimization problems. This section builds this generic mathematical representation from regular optimization, to multi-level optimization, and then through unique differentiating features, to SoS optimization problems.

Multilevel optimization was introduced in 1952 by von Stackelberg who proposed a two level strategy for use in systems where top level policy changes influences decisions

at the lower level. [54] We first describe the difference between well known optimization formulations, and how adding a few specific features to this standard formulation may be useful in representing *SoS*-optimization problems. We then formalize our discussion and offer a mathematical description of this framework (Section 2.3.1).

Characteristics of a conventional parameter optimization problem can be described using the generic formulation shown in Equation 2.13. Typically, an optimum point ($\mathbf{x}^*$), such that $x \in \mathbb{R}^n$ is to be found with respect to an objective function f or a vector of objective functions $\mathbf{J}$. The problem may be further restricted by variable bounds (LB, UB), equality (H) and inequality (G) constraints. In this problem description, the size of the design variable is fixed, that is, $\mathbf{x} \in \mathbb{R}^n$ is true for all time / all generations / all optimization iterations.

$$
\begin{aligned}
&\text{Min/Max} && \mathbf{J}(\mathbf{x}) = [f_1, f_2, \ldots, f_k]^T(\mathbf{x}) && : k \geq 1 \\
&\text{Subject To} \\
&\text{Inequality} && \mathbf{G}(\mathbf{x}) = [g_1, g_2, \ldots, g_l]^T(\mathbf{x}) \leq \mathbf{0} && : l \geq 0 \\
&\text{Equality} && \mathbf{H}(\mathbf{x}) = [h_1, h_2, \ldots, h_m]^T(\mathbf{x}) = \mathbf{0} && : m \geq 0 \\
&\text{Variable Bounds} && \mathbf{LB} \leq \mathbf{x} \leq \mathbf{UB} \\
&\text{Where } \mathbf{x} \in \mathbb{R}^n && : n \geq 1
\end{aligned}
\tag{2.13}
$$

We may modify this description to yield a 'multi-level' optimization problem with s number of levels (Equation 2.14). Each upper level in the list of s levels uses variables that are optimum values of the lower levels. The symbols f, g , h , $\mathbf{J}$ etc. have their usual meanings, however a subscript is added to identify the current level of operation. Also, we notice that although the number of equality and inequality constraints may be zero, there must be a top level objective function J in each level. In other words, multiple layers of unconstrained optimization functions form a valid multi-level, multi-objective problem.

$$
\begin{aligned}
&\text{Min/Max} && J^1(\mathbf{x}^1) = [f_1^1, f_2^1, \ldots, f_k^1]^T(\mathbf{x}^1) && : k^1 \geq 1 \\
&\text{Subject To} \\
&\text{Inequality} && G^1(\mathbf{x}^1) = [g_1^1, g_2^1, \ldots, g_l^1]^T(\mathbf{x}^1) \leq \mathbf{0} && : l^1 \geq 0 \\
&\text{Equality} && H^1(\mathbf{x}^1) = [h_1^1, h_2^1, \ldots, h_m^1]^T(\mathbf{x}^1) = \mathbf{0} && : m^1 \geq 0 \\
&\text{Variable Bounds} && \mathbf{LB}^1 \leq \mathbf{x}^1 \leq \mathbf{UB}^1 \\
&\text{Where } \mathbf{x}^1 \in \mathbb{R}^{n^1} && : n^1 \geq 1
\end{aligned}
\tag{2.14a}
$$

Where $\mathbf{x}^1$ solves:

$$
\begin{aligned}
\mathbf{x}^1 = &\text{Min/Max} && J^2(\mathbf{x}^2) = [f_1^2, f_2^2, \ldots, f_k^2]^T(\mathbf{x}^2) && : k^1 \geq 1 \\
&\text{Subject To} \\
&\text{Inequality} && G^2(\mathbf{x}^2) = [g_1^2, g_2^2, \ldots, g_l^2]^T(\mathbf{x}^2) \leq \mathbf{0} && : l^2 \geq 0 \\
&\text{Equality} && H^2(\mathbf{x}^2) = [h_1^2, h_2^2, \ldots, h_m^2]^T(\mathbf{x}^2) = \mathbf{0} && : m^2 \geq 0 \\
&\text{Variable Bounds} && \mathbf{LB}^2 \leq \mathbf{x}^2 \leq \mathbf{UB}^2 \\
&\text{Where } \mathbf{x}^2 \in \mathbb{R}^{n^2} && : n^2 \geq 1
\end{aligned}
\tag{2.14b}
$$

$\vdots$

Where $\mathbf{x}^{s-1}$ solves:

$$
\begin{aligned}
\mathbf{x}^{s-1} = &\text{Min/Max} && J^s(\mathbf{x}^s) = [f_1^s, f_2^s, \ldots, f_k^s]^T(\mathbf{x}^s) && : s^1 \geq 1 \\
&\text{Subject To} \\
&\text{Inequality} && G^s(\mathbf{x}^s) = [g_1^s, g_2^s, \ldots, g_l^s]^T(\mathbf{x}^s) \leq \mathbf{0} && : l^s \geq 0 \\
&\text{Equality} && H^s(\mathbf{x}^s) = [h_1^s, h_2^s, \ldots, h_m^s]^T(\mathbf{x}^s) = \mathbf{0} && : m^s \geq 0 \\
&\text{Variable Bounds} && \mathbf{LB}^s \leq \mathbf{x}^s \leq \mathbf{UB}^s \\
&\text{Where } \mathbf{x}^s \in \mathbb{R}^{n^s} && : n^s \geq 1
\end{aligned}
\tag{2.14c}
$$

Authors have utilized existing methods from optimization, Multi-disciplinary optimization (MDO), Analytical Target Cascading (ATC) and decomposition based approaches to formulate SoS optimization problems as nested or hierarchical versions of regular, multi-level optimization problems. [20, 21, 55–57] Others have solved application problems (especially simultaneous, bi-level, aircraft and fleet optimization problems) that are posed as SoS design optimization problems. [55, 58–61] As detailed in [62], these frameworks generally consider a static design space, with levels of optimization problems with a required top-level objective, and a single level of abstraction. Here we present additional features present in SoS optimization problems that distinguish them from conventional multi-level optimization problems.

For explanation purposes, let us re-introduce the targeted SoS optimization application problem that involves the selection of suitable UAVs for imaging a plot of agricultural land ('Precision Agriculture Swarm'). The problem span three levels - The sub-system level contains parts of a UAV such as engines, wings, tail sections etc.; the system or aircraft level containing a library of aircraft assembled (optimally or feasibly) from the sub-system level; and an SoS level or Swarm level containing networks formed by selecting instances from the aircraft library. We shall use this example to explain features of the full fledged SoS optimization problem that is obtained by modifying the standard multi-objective, multi-level problem shown in Equation (2.14).

- *Optional* objective function J for any given level (1 to s)

> Unlike conventional multi-level problems, an SoS problem need not necessarily optimize a function J at all levels. Thus, it may be sufficient to have a feasible x at a particular level, rather than optimizing for a minimum or maximum x as in constraint programming. Alternatively, a level may also have an unconstrained optimization problem (only J with no f or g). In our intended application problem (Precision agriculture UAV swarm), the top level SoS may only require that the selection of UAVs (x) completes a mission (representing a *feasible* design

point). The SoS may not be optimal; that is, the mission may not be completed in minimum time or with maximum area of coverage. If a method for measuring how well the capability is satisfied exists (through some metric, for example, probability of mission success), this may be treated as an objective function. We then have a measure of under- or over-achievement with respect to some desired level of performance.

- Evolving design space

The size of the design variable vector (n) and the objective function are fixed in conventional optimization problems, but may vary (in a predictable fashion) for SoS optimization problems. We must further understand the nature of this evolution. In our application problem, as the number of UAVs in the aircraft level of the SoS optimization problem increases, the number of potential SoS (or network of UAVs) increases tremendously.

- Uncertain resources

Solutions and performance of an optimization must remain relatively unchanged when exposed to uncertain conditions. Uncertainties may exist in 1. Design Variables, 2. Environmental parameters, 3. Objective function or 4. Constraints. Uncertainty in a particular item described here may be *aleatory* (intrinsic and irreducible kind) or *epistemic* (due to lack of knowledge a designer may have). In our application problem, weather, policy changes, manufacturing limits and operating conditions are all uncertain.

- Multiple levels of abstraction

A single design variable represents one feature, but possibly at multiple levels of abstraction. In other words, a design variable that represents the same feature may have varying levels of detail or fidelity. For example, imagine that a particular aircraft (at the systems level in the optimization problem) is a design variable for the SoS level. So, the SoS may 'choose' this aircraft to form one of its nodes. The aircraft (a design variable) may be represented with increasing levels of detail. Two extreme examples are listed below :

1. Wing Area
2. CAD model of entire wing, fuselage and stabilizers along with engines and other appendages

- Choice-set varies with time

In a real-world SoS, systems may retire or may be replaced by better systems ahead of their scheduled time for end of service. In order to incorporate this into the generic formulation of the SoS optimization template, we introduce swapped variables y_c at each level. These variables replace the values of a design variable vector x, thereby inducing no change in the size of the design space n. Progress in time may be incorporated by changing the simulation environment or tweaking simulation parameters. In our application problem, imagine that a particular engine is to be replaced by a newer version of the same engine on all aircraft that use it, but after roughly 100 missions. The sub-system library (represents the set of all x in the sub-system level) then replaces the older engine (retired) x_c with a newer engine y_c.

- Capability flowdown by setting targets

The framework must involve flow of information in the

1. Upward or conventional multi-level optimization direction i.e. optimum values or products from lower levels are used in upper levels
2. Downward direction, i.e. higher levels provide feedback on as to how effective the 'optimum' values suggested by the lower levels were through targets.

For example, an airfoil may be optimized for maximum $\frac{cl}{cd}$ at a lower level, but may not necessarily be selected to form a rib on an optimum wing on the upper level. The target specification aims to establish communication in the downward direction. Targets can be made part of objectives in each level. For example, a previously found optimum objective function value, say T can be improved by using $\min f - T$ in the approach. T may also indicate customer needs and/or market requirements.

- Expert in the loop

Since no realistic SoS optimization can be made intelligent enough to be completely automated, provisions must be made to accommodate human decisions and to allow for the morphing of objective functions, variable dimensions or constraints during the implementation of the framework for SoS optimization. For example, an expert may *know* through intuition or an external analysis that a particular type of UAV must not be considered for the current problem. Formulating and reformulating the SoS problem to obtain a high quality, refined result must also be an easy, efficient process.

2.3.1 Mathematical Formulation

Although the changes made are subtle, they complicate or prohibit use of conventional optimization frameworks and algorithms used for standard multilevel or multi-objective problems. The final framework that incorporates all the aforementioned details is given in equation (2.16). In Equation (2.16), superscripts 1 to s represent the level number. In a three level SoS optimization problem, the top-most level is usually referred to as the SoS level, whereas lower levels are called system and sub-system levels. For problems with more than three levels, the level number (1 to s) is used as is for identification, with level 1 representing the top-most (or SoS level), and level s representing the lowest level. Depending on various factors, one can decide as to which level of fidelity (j) is required for an optimization step (iteration) i at that time (t) being simulated in the model. Also, variables at a particular level may be swapped with new ones ($x \bowtie y_c$), may increase ($\mathbf{x}(t,i+1) = x_{new} \cup \mathbf{x}(t,i)$) or decrease in dimension ($\mathbf{x}(t,i+1) = \mathbf{x}(t,i) \setminus x_{old}$). To handle uncertainties, we use features from the *Robust Counterpart Approach* [63] Suppose we wish to optimize a function f over a variable set $\mathbf{x}$. Let $\mathbf{x}$ vary in precision / confidence by ε, and due to environmental uncertainties / operational conditions (α). Then,

$$\text{Min/Max } f(x|\varepsilon,\alpha) \equiv \sup_{\xi \in (x|\varepsilon)} / \inf_{\xi \in (x|\varepsilon)} f_R(\xi,\alpha) \tag{2.15}$$

where f_R is the robust counterpart of f. Constraint functions may also be represented in the same manner. Adding an expert in the loop, and implementing capability flowdown based on target specification are programmatic details that must be incorporated while actually implementing the optimization framework, and are problem specific. An objective function in a level s may be a combination of other simpler functions. That is, $f^s = \sum \hat{f}^s$. We consider the case when f^s evolves to include another new function in the sum (i.e., $f^s = \sum \hat{f}^s + \hat{f}_{new}$), especially when $\hat{f}$ are all convex and Lipchitz continuous (see chapter 5 for details). To enable capability flowdown, we use targets as a part of objectives in each level (an objective f is replaced by $f - T$ to measure under-achievement or over-achievement). For example, a previously found optimum objective function value, say f^*

can be improved by minimizing $f - f^*$ in an attempt to find a design with an objective function that is better than the previously found f^*. The quantity f^* here may also indicate customer needs and/or market requirements. However, textual targets and customer requirements, as they are commonly presented, may have to be quantified before use in the framework. For the final mathematical stencil, see Equation 2.16.

$$
\begin{aligned}
&\text{Min/Max :} \\
&\mathbf{J^1_R}(\mathbf{x}^1|\boldsymbol{\varepsilon}^\mathbf{1},\alpha) \equiv \sup_{\xi^1 \in (x^1|\varepsilon^1)} / \inf_{\xi^1 \in (x^1|\varepsilon^1)} J^1(\xi^1) = [f_1^1, f_2^1, \ldots, f_k^1]^T(\xi^1) \\
&\text{Subject To} \\
&\mathbf{G^1_R}(\mathbf{x}^1|\varepsilon^1,\alpha^1) \equiv \sup_{\xi^1 \in (x^1|\varepsilon^1)} / \inf_{\xi^1 \in (x^1|\varepsilon^1)} G^1(\xi^1) = [g_1^1, g_2^1, \ldots, g_l^1]^T(\mathbf{x}^1) \leq \mathbf{0} \\
&\mathbf{H^1_R}(\mathbf{x}^1|\varepsilon^1,\alpha^1) \equiv \sup_{\xi^1 \in (x^1|\varepsilon^1)} / \inf_{\xi^1 \in (x^1|\varepsilon^1)} H^1(\xi^1) = [h_1^1, h_2^1, \ldots, h_m^1]^T(\mathbf{x}^1) = \mathbf{0} \\
&\mathbf{LB} - \boldsymbol{\varepsilon}^1 \leq \mathbf{x}^1 \leq \mathbf{UB} + \boldsymbol{\varepsilon}^1 \\
&\text{Swap Variable values : } (x|\varepsilon)_c^1 \bowtie (y|\upsilon)_c^1 \\
&\text{Evolve Design Space :} \\
&\mathbf{x}^1(t,i+1) = x_{new}^1 \cup \mathbf{x}^1(t,i) \parallel \mathbf{x}^1(t,i+1) = \mathbf{x}^1(t,i) \setminus x_{old}^1 \\
&f_p^1 = \sum_q \hat{f}_{pq}^1 + \hat{f}_{new}^1,\ \forall p \in [1,k^1], q \geq 1 \\
&\text{Where : } \mathbf{x}^1 \in \mathbb{R}^{n^1} \\
&k^1 \equiv k^1(t,i,j) \geq 0 \\
&l^1 \equiv l^1(t,i,j) \geq 0 \\
&m^1 \equiv m^1(t,i,j) \geq 0 \\
&n^1 \equiv n^1(t,i,j) \geq 1
\end{aligned}
\tag{2.16a}
$$

$\vdots$

Where ξ^{s-1} solves:

$$
\begin{aligned}
&\text{Min/Max} \\
&\mathbf{J_R^s}(\mathbf{x}^s|\boldsymbol{\varepsilon}^\mathbf{s},\alpha) \equiv \sup_{\xi^s \in (x^s|\varepsilon^s)} \Big/ \inf_{\xi^s \in (x^s|\varepsilon^s)} J^s(\xi^s) = [f_s^1, f_s^1, \ldots, f_k^s]^T(\xi^s) \\
&\text{Subject To} \\
&\mathbf{G_R^s}(\mathbf{x}^s|\varepsilon^s,\alpha^s) \equiv \sup_{\xi^s \in (x^s|\varepsilon^s)} \Big/ \inf_{\xi^s \in (x^s|\varepsilon^s)} G^s(\xi^s) = [g_1^s, g_2^s, \ldots, g_l^s]^T(\mathbf{x}^s) \leq \mathbf{0} \\
&\mathbf{H_R^s}(\mathbf{x}^s|\varepsilon^s,\alpha^s) \equiv \sup_{\xi^s \in (x^s|\varepsilon^s)} \Big/ \inf_{\xi^s \in (x^s|\varepsilon^s)} H^s(\xi^s) = [h_1^s, h_2^s, \ldots, h_m^s]^T(\mathbf{x}^s) = \mathbf{0} \\
&\mathbf{LB} - \varepsilon^s \leq \mathbf{x}^s \leq \mathbf{UB} + \varepsilon^s \\
&\text{Swap Variable values : } (x|\varepsilon)_c^s \bowtie (y|\upsilon)_c^s \\
&\text{Evolve Design Space :} \\
&\mathbf{x}^s(t,i+1) = x_{new}^1 \cup \mathbf{x}^s(t,i) \parallel \mathbf{x}^s(t,i+1) = \mathbf{x}^s(t,i) \setminus x_{old}^s \\
&f_p^s = \sum_q \hat{f}_{pq}^s + \hat{f}_{new}^s, \ \forall p \in [1,k^s], q \geq 1 \\
&\text{Where } \mathbf{x}^s \in \mathbb{R}^{n^s} \\
&k^s \equiv k^s(t,i,j) \geq 0 \\
&l^s \equiv l^s(t,i,j) \geq 0 \\
&m^s \equiv m^s(t,i,j) \geq 0 \\
&n^s \equiv n^s(t,i,j) \geq 1
\end{aligned}
\tag{2.16b}
$$

2.3.2 Boundaries of Applicability

Although the formulation shown in Equation (2.16) is generic and widely applicable, there are limitations. For example, the only kind of uncertainties that can be handled are deterministic (defined by fixed parameters ε and α) in nature. Beyer and Sendhoff identify two more forms of uncertainty - *probabilistic* (involving likelihood of by which a certain

event occurs) and *possibilistic* (fuzzy measure describing if an event is plausible or believable).

There do exist SoS frameworks that address other dimensions of the rapidly growing SoS field like complexity, simulation [64], capability [65], architecting and exploration [66]. However a specialized mathematical framework for optimizing SoS does not exist. This framework is set up to handle these SoS *ilities* in the form of objective functions (J) or constraints (G or H) and focuses on optimization rather than measuring a particular metric. As such, the framework is metric-independent. That is, other frameworks can be effective add-ons used to expand the current framework. We imagine our framework to resemble a tree of MDO problems (See figure 2.4), involving inter- and intra-level interactions and evolving design spaces, with information, data or target instructions flowing in the upward or downward directions. The individual MDO problems (gray boxes) are related through the products or objects they design as a result of the optimization run. Products of lower levels (or sub-systems) are assembled at higher levels to form larger products (or systems). Each level designs a product, and may involve multiple disciplines and fidelities. The products from lower levels that are part of an assembly in the higher level are transferred appropriately. The problem can then be (roughly) imagined as a combination of Analytical Target Cascading (ATC) and MDO, minus the additional features described in Section 2.3.

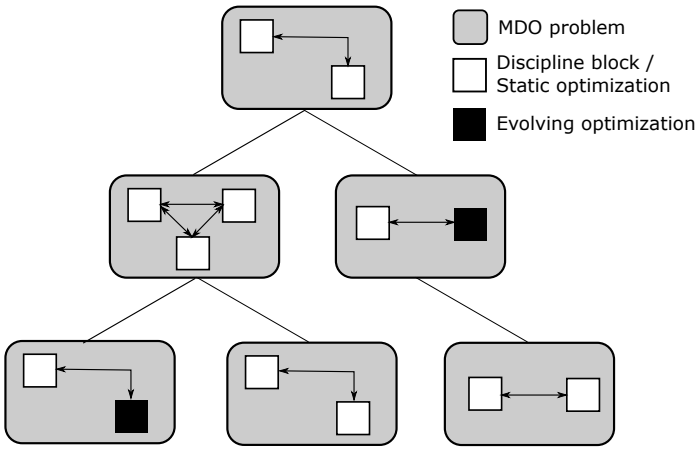


Figure 2.4.: Framework aims to solve a tree of MDO problems where products are designed at each level (sub-system, system and SoS) and appropriately transferred to higher levels when required

2.4 Evolving design spaces through Numerical Continuation

Evolving design spaces are an apposite characteristic of System of Systems (SoS). Although 'evolving nature' is portrayed as one of the most important traits of a SoS (DeLaurentis *et al.* [67]), the context of the word *evolving* was not the same as the one adopted in this work. Here, our aim is to formalize the notion of SoS optimization by providing a mathematical stencil, boundaries of applicability, suitable optimization algorithms, typical application problems and useful extensions to the overall framework. From the perspective of optimization, SoS may evolve over time. This form of evolution is more suitable for use in Dynamic Programming environments. On the other hand, the resources and operations of the SoS can be optimized for *a* particular time instant, range of times, or for all times. This form is more suited to multi-level optimization. The mathematical basis of this framework for SoS optimization is inspired by the field of multilevel optimization, but only resembles it in terms of overall form and structure. We demonstrate the use of evolving design spaces using two solution strategies as part of the two application problems (Chapters 4 and 6).

Evolving design spaces are rarely used in the widely available literature on optimization. A design space may be finite or infinite. Imagine a grid of cells that can be individually activated to add them to the set of design variables $\mathbf{x}$, or individually switched off to remove them from the active set of variables. This design space is evolving but finite. This kind of design space evolution is relatively more common in structural topology optimization. Jang and Kwak use design space adjustment based on a fixed grid and demonstrate the superiority of their method over conventional topology optimization. They report lesser computational cost and an increased chance of finding global optima incrementally. [68] To handle the possible discontinuities that arise in objective functions and constraints due to changing design space, Kim and Kwak propose a design continuation method that is also used in this work. [69] Here, the existing design points are represented with the new set of design variables.

In the above examples of structural topology, the evolving design space is based on a fixed background grid. Thus the maximum resolution or fidelity is fixed. On the other hand, design spaces may be infinite (no maximum resolution). Padmanabhan *et al.* use a modified branch-and-bound technique for acceleration (optimization) of streaming applications (video / data streaming online etc.). [70] To contain the search in an infinite space, the authors use heuristics to order the variables created (called branching variables) such that each branching leads to maximum decomposition of the search space. Zahir and Zheengong demonstrate that a diminishing design space may also be useful. In their experiment that involves multiple fidelity objective functions, a Genetic Algorithm based solver finds an optimum of the lower fidelity function space. [71] Then, a surrogate model is built around the 5% region around the low fidelity optimum. Thus, although the design space evolves (reduces), the number of possible reductions are infinite.

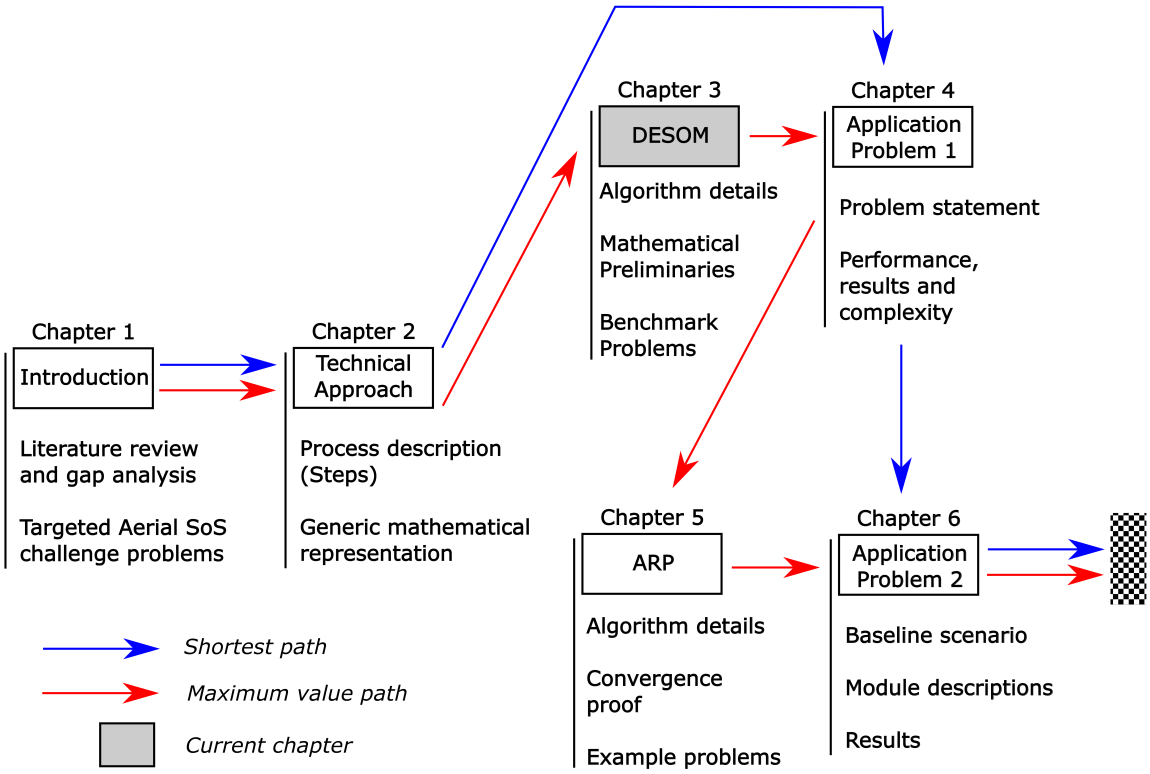
A primary contribution of this thesis is the development of a provably convergent method to solve optimization problems with evolving design spaces. This method, Adaptive Random Projections (ARP) will be discussed before we solve the second application problem (in chapter 5).

2.5 Nature of Application Problems

Our two demonstration problems are fundamentally different, but use the same optimization framework to find a solution across multiple levels. However, only algorithms and techniques that are relevant to specific problem features are used. The table 2.1 classifies the two problems, and also paves the way for description of these algorithms and solution techniques in the following chapters.

Feature	**Application no.**		**Solution Method**	**Relevant Chapters**
	1	**2**		
No. of levels	3	3	SoS Optimization framework	2
Inter-level interactions (apart from product transfer)	No	Yes	-	6
Intra-level interactions	Yes	Yes	-	4 and 6
Intractable black-box Optimization	Yes	Yes	DE-SOM and DE-SOM2	3, 4 and 6
Multi-disciplinary Optimization	No	Yes	Enhanced Collaborative Optimization, Individual Discipline Feasible	6
Multi-fidelity Optimization	No	Yes	Value of Information (VoI)	6
Parameter Uncertainty	Yes	No	Robust Counterpart approach	2 and 4
Sensitivity analysis	No	Yes	Random sampling	6
Evolving design space: Type 1	Yes	No	Numerical Continuation	4
Evolving design space: Type 2	No	Yes	Adaptive Random Projections	5 and 6
Time budget enforced/ Process and product complexity monitored	Yes	No	Hierarchical Complexity	4
Omnipotent Central Authority	Yes	No	Single Process Multiple Data (SPMD)	4
Mediating Central Authority	No	Yes	Sequential programming in parallel instances	6

Table 2.1: Comparison of Application problem 1 and 2



Differential Evolution with Self Organizing Maps (DESOM) forms one of the backbones of our framework, and is the preferred evolutionary algorithm used as the optimizer in various levels of the application problems described in Chapters 4 and 6.

3. Differential Evolution with Self-Organizing Maps (DE-SOM and DE-SOM2)

This chapter describes a novel hybrid Differential Evolution (DE) Self-Organizing-Map (SOM) Algorithm for the optimization of expensive black-box functions. Several improvements are made to DE-SOM (DE-SOM2) to solve the IEEE CEC benchmark set. We recognize that our framework involves the use of several computationally expensive black-box tools that need to be used for multi-fidelity optimization of aircraft structures, aerodynamics and propulsion systems. Noting that our overall SoS optimization problem could possibly involve several black-box functions, this algorithm will be a good candidate algorithm for exploring solutions at all three levels of the Aerial SoS.

Typical objective functions arising from results (function calls) of these tools could represent a multi-modal, rough, discontinuous landscape. Thus, it is suggested that one uses evolutionary algorithms (see section 3.1 below). Another advantage of evolutionary optimization algorithms that makes it suitable for use in our framework is that at any given time, once can record the best solution obtained so far, which is associated with several other designs (depending on the population size) that may also be feasible for use in higher levels of the problem. This is how the designs are presented as "flexible" ones, rather than singular points in the design space. Evolutionary algorithms, however, are expensive by nature. This is our attempt to modify an existing evolutionary algorithm, with the motivation of reducing the total number of function evaluations. Once the DE-SOM algorithm is validated with benchmark functions, sample application problems are solved.

3.1 Introduction to DE-SOM

There has been significant interest in the use of evolutionary methods for the global optimization of real-valued black-box problems for which analytical methods are not ap-

plicable. Differential Evolution (DE) is a parallel direct optimization algorithm which was recently introduced by Storn and Price. [72] Vesterstrom *et al.* showed that DE performed better than other evolutionary algorithms such as Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) on a suite of 34 widely used benchmark functions. [73] Other authors confirmed similar results when DE was pitted against time-tested global methods such as Annealed Nelder Mead Approach (ANM), the Breeder Genetic Algorithm (BGA), the EASY (Evolutionary Algorithm with Soft Genetic Operators) algorithm, the method of Stochastic Differential Equations, Cuckoo-search, Artificial Bee Colony and interactive GA algorithms. [74–76] Common non-hybrid variants of DE like *DE/rand/1/bin* and *DE/best/1/dir* provide only marginal increase in performance for the same number of function evaluations. For the interested reader, these are presented by Mezura-Montes *et al.* [77] Here, we use an artificial neural network variant to enhance the DE algorithm.

The Self-Organizing Map (SOM) is the most popular artificial neural network algorithm in the unsupervised learning category. [78] It is often associated with clustering, data visualization, dimensionality reduction, nonlinear data projection and manifold mapping. [79] The SOM uses a network of neurons to form a discrete topological mapping of a set of input vectors. In our implementation, we use the positions of the DE population members as 'inputs' to the SOM. The position of a neuron, also known as its 'weight', w_i, is a vector that has a dimension equal to that of any input vector in the DE population ($X_i \in \mathbb{R}^n$). The weights of the neurons in this SOM are updated until they converge to the input vectors (X_i). Neurons of the SOM network move towards clusters of the DE population members. As such, once the SOM structure 'converges', it may be associated with one or more DE population members. The number of associations of a particular neuron in the SOM is called 'hits'. The SOM network morphs and moves to alter the "focus region" since the network itself is a result of the positions of the previous generation's population members.

DE, PSO and other evolutionary algorithms have been commonly used to evolve the weights of SOM for performance enhancement, and this has been the only form of hybridization attempted. [80–84] We have used a converse form of hybridization wherein the global optimizer is enhanced by SOM. Obayashi and Sasaki's work on data mining of

Pareto solutions involving more than three objectives is also an interesting application of the SOM algorithm in optimization. [85] In their work, edges of the SOM of the objective function values (obtained from the result of a Multi-Objective Genetic Algorithm (MOGA) run) is a representation of a 4D Pareto front. Furthermore, the authors cluster design variables based on another SOM to help visualize design variable trade-offs. Here we present a novel method to improve the performance of DE by introducing a learning phase that helps contain the likely optimum, while continuously reducing the active design space without the use of additional function evaluations. In section 3.2, we offer a qualitative discussion of the DE-SOM algorithm and compare it to the regular DE algorithm. Then in Section 3.3, we show mathematically how DE-SOM may achieve an optimal solution with lesser number of total function evaluations. Section 3.4 supports this claim by comparing experimental results of 15 widely used benchmark functions. Finally, we compare the DE-SOM algorithm with GA and DE for an airfoil optimization problem (section 3.5). Versions of this airfoil optimization problem are used in both the application problems (chapters 4 and 6).

3.2 The DE-SOM Algorithm

Consider a fitness function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Both, the DE and DE-SOM algorithm first form a mutant vector v_i from three mutually exclusive vectors x_{r1}, x_{r2} and x_{r3} chosen from a total population consisting of NP members. The crossover probability CR decides if the trial vector u_i inherits components of the original vector x_i or the mutated vector v_i in that generation G. The primary difference between the DE and DE-SOM algorithms occurs in the selection step (see Algorithms in Appendix A). In DE, the function value of the trial vector is compared with that of the original vector $\big(f(u_{i,j}) \leq f(x_{i,j})\big)$ to decide which vector wins a position in the population of the next generation. DE-SOM uses a random variable p (called the pswitch value) to control the amount of function evaluations saved. A value of $p = 1$ implies that the algorithm used is purely DE, and a value of $p = 0$ means that the DE-SOM algorithm will be used for all vectors in that generation. In our research,

p values ranged from 0.25 to 0.75. An SOM is used to construct the convex hull of the population in the current generation, and the convex hull of the current generation is used for testing the presence of a new trial vector near the likely optimum.

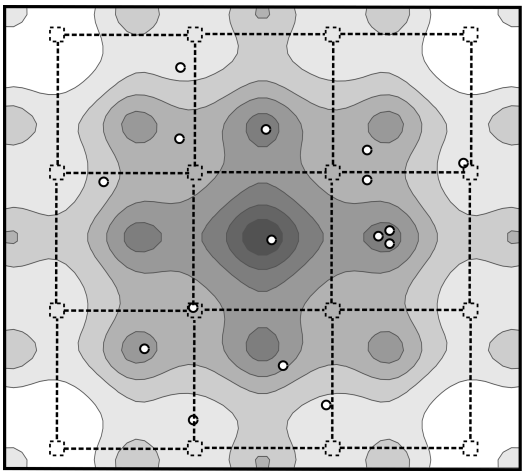


Figure 3.1.: The DE population members (white circles) are scattered randomly on a fitness landscape with one global optima (center) and several local optima.The SOM (dashed) is initialized as a regular 4×4 grid.

The SOM is a network of neurons whose positions are given by weights $w_i \in \mathbb{R}^n$. The weights of the neurons are also n-dimensional since they are used in the elite replacement step in which a valuable neuron (one that may have a favorable function value) replaces a low scoring population member. In a given generation, the members of the population x_i occupy different positions on the fitness landscape of the function being minimized (see white circles in figure 3.1). A 4×4 network of neurons (or SOM template) is shown in figure 3.1. Neurons of this SOM are represented by dashed boxes, and links between them by dashed lines. The neurons of the SOM change their weights using an unsupervised learning algorithm until convergence (note that this 'convergence' of the network is different from the convergence of the optimization algorithm. This step occurs at every generation of the DE-SOM algorithm). At convergence, the network forms a closed polygonal volume with vertices that coincide with the extremum members of the population (see figure 3.2). Figure 3.2 also highlights some salient points of the learning step of this algorithm: Point '1' shows that the outermost neurons in the SOM *snap* onto the extremum members of the DE

population. While most neurons are associated with one or more DE population members, some neurons may be unassociated (or may have zero *hits*). Point '2' shows that the SOM itself may not be convex in some regions, but the convex hull of the extremum points of the SOM always contains (or is identical to) the convex hull of the DE population members. Point '3' shows that a particular neuron may be associated with multiple DE population members (in this case, three).

Like any other evolutionary algorithm, DE tends to form clusters around local or global optima as the generation number progresses (when approaching convergence). Thus, a neuron associated with multiple population members (in other words, a neuron with a large number of hits) is more likely to have a favorable function value. In the elite replacement step, a fixed percentage of low scoring DE population members are replaced with neurons with a large number of hits. Point '4' indicates that the global optimum is contained in the convex hull for all subsequent steps if a trial vector happens to be placed there after a particular intermediate step, as in other evolutionary algorithms. The difference in the selection step helps save unnecessary function evaluations. Suppose the DE population member at point '2' has a poor function value. A regular DE algorithm would generate a trial vector, and compare function values of the original vector and the new trial vector (two function evaluations). DE-SOM on the other hand, would generate a trial vector, but would test if the trial vector is contained in the convex hull of the converged neuron weights (positions) of the previous generation. In our hypothetical scenario, let us assume that the trial vector u_i is inside the convex hull of the SOM. During the initial stages when exploration is premature, there is no guarantee that

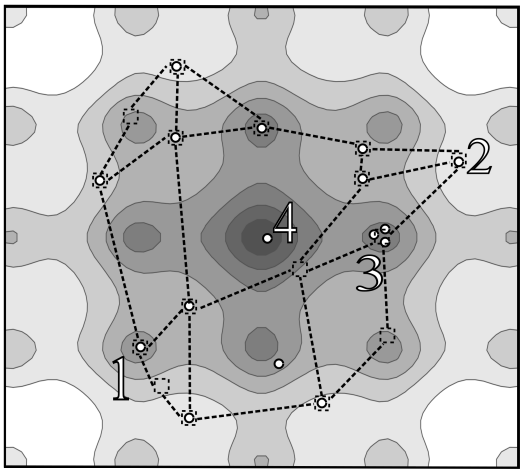


Figure 3.2.: The SOM neurons(dashed boxes) converge to the members of the DE population (white circles) at the end of each generation.

a convex hull test is equivalent to a function value comparison test. However, as will be shown in the mathematical treatment of the algorithm, the mean value of the positions of the population members are representative of the point with the best function value. This is always true during the final stages of convergence when all the population members may be assumed to reside on a smooth, continuous function slope towards the optimum. It cannot be stressed further that the SOM simply assists the search by preventing function evaluations, but the primary exploration is still performed by DE (or any other evolutionary algorithm that SOM is applied to). The elite replacement step also introduces valuable members to the original DE population at no additional functional cost. The next section formalizes our discussion of the DE-SOM algorithm, and provides bounds of the parameter p that is crucial in deciding the number of function evaluations saved.

3.3 Convergence and Parameter Bounds

In this section, we provide mathematical preliminaries that better explain the DE-SOM algorithm. The aim of this section is to compare the DE and DE-SOM algorithm, and to say that the DE-SOM algorithm achieves the same Expected Value of the Trial Vector (see section 3.3.1) and comparable velocity of the population members (section 3.3.2). The variation in population is derived as a function of the probability of containment q in section 3.3.3. Finally we find critical values of this function and discuss how a member x_i can be converted to a trial vector u_i that is most probably contained in the convex hull of the members of the current generation. Note that only partial derivations / final results will be provided here. For detailed derivations, kindly refer to Appendix B.

3.3.1 Expected Value of the Trial Vector (u_i)

Let x_i be the i'th population member in the final stages of the DE-SOM algorithm. We assume that all the members of the population are concentrated around the optimum. For the *DE/rand/1* version of the algorithm, the mutant vector v_i is generated as follows:

$$v_i = x_{r1,i} + F \times (x_{r2,i} - x_{r3,i}) \tag{3.1}$$

where F is the mutation scaling factor and the x_{rk} terms with $k \in \{1,2,3\}$ are mutually exclusive random vectors drawn without replacement from the population of the current generation. Since the vectors x_{rk} are independent of each other, $P(x_{ri} \mid x_{rj}) = P(x_{ri})$. The trial vector u_i is a result of the crossover operation given by

$$u_i = \begin{cases} v_i & \text{if } rand \leq CR \vee j = j_{rand} \\ x_i & \text{otherwise} \end{cases} \tag{3.2}$$

The expected value of the trial vector u_i is calculated in equation 3.3 (see Appendix B for details).

$$E(u_i) = (1 - CR) \cdot (x_i) + CR \cdot (x_{av}) \tag{3.3}$$

Equation 3.3 implies that for values of $CR \in [0,1]$, the expected value of the trial vector u_i (after mutation and crossover) lies in between x_i and x_{av}. Equation 3.3 is true even for the DE algorithm (without the SOM), but we derive it here to highlight the following. Since the convex hull of a set of vectors includes the average vector, we can conclude that the best vector of the current generation is close to the average of the convex hull. A pure DE algorithm would naturally progress towards the average of the population. We achieve the same effect by containing the average vector, and replacing badly performing extremum points with internal neurons which are more likely to be optimal (i.e. neurons with a large number of hits). Since $CR \leq 1$, points are forced to be on or inside the convex hull of population members in DE. For instance, point '2' in figure 3.2 will move towards the average of the population members (around point '4') in the DE algorithm. In DE-SOM, the same point '2' will move towards '4' due to either reduction of the convex hull in that direction, or

elite replacement of the poorly performing point '2', with the neuron above point '3'. Thus we can effectively replace the function comparisons in *some* generations with the convex hull test (for example, the Quickhull algorithm) to achieve the same expected value of a population member in the next generation. [86, 87]

3.3.2 Desired velocity of population members

The expected value of the velocity of a vector x_i (with respect to time or iteration number t) is derived in Appendix B and can be given by equation 3.4

$$\begin{aligned} E(\frac{dx_i}{dt}) &= \frac{1}{2}E(u_i - x_i) \\ &= \frac{1}{2}(E(u_i) - E(x_i)) \\ &= \frac{1}{2}((1-CR)\cdot(x_i) + CR\cdot(x_{av}) - x_i) \\ &= \frac{CR}{2}(x_{av} - x_i) \end{aligned} \tag{3.4}$$

Since the expected value of the trial vector u_i is simply a function of the individual operators for mutation and crossover (as derived in section 3.3.1), we are able to get an expression for expected velocity of a member that is similar to DE. Therefore from equation 3.4, close to convergence, the velocity of each member of the population is directed towards the average of the population. For any non-empty convex set A, a point $a \in A$ is an extremal point if $A - \{a\}$ is also convex. [88] Here, we are replacing the extremal points (see point '2' in figure 3.2) with more optimal internal points. These new internal points are added to existing groups or clusters in the elite replacement step. This aids in the clustering of the members around optimal regions in the function space. Thus adding new members to existing clusters during the elite replacement step accelerates movement towards the average. Here, we have achieved the desired velocity of population members without the use of function evaluations.

3.3.3 Variation of the population

Here we discuss the benefits of DE-SOM by calculating the number of function evaluations saved, and by comparing the variance of the expected value of vectors after the mutation and crossover steps for DE and DE-SOM. Our aim in this section is to show that comparable variation in population is obtained in DE and DE-SOM. Let C be the total number of generations in the case of DE and DE-SOM, and p be the switching probability of using either DE-SOM or DE for the current generation. Let NF be the number of function evaluations, and NP be the number of population members. For DE, it can be easily verified that

$$NF_{de} = 2C \cdot NP \tag{3.5}$$

whereas for DE-SOM, only the generations that use pure DE will incur function evaluations. Let C_{de} be the generations that use DE, and C_{desom} such that $C_{de} + C_{desom} = C$, and therefore $p = \frac{C_{de}}{C}$. Then :

$$\begin{aligned} NF_{desom} &= (2C_{de}NP) + 2C_{desom}NP \cdot 0 \\ &= 2pC(NP) \end{aligned} \tag{3.6}$$

Since $p \leq 1$, $NF_{desom} \leq NF_{de}$. Note that p may not be perfectly random, and hence equation 3.6 is only valid at a large number of generations. Now, saving function evaluations at the cost of quality of the solution is undesirable. Therefore, we need to study the variance of the positions of the population members. We derive this quantity in Appendix B, and refer to it below.

$$\begin{aligned} E(Var(u)) &= \left(\frac{q^2}{NP} + \left(2F^2 - \frac{2}{NP} \right) q + 1 \right) Var(x) \\ &= L_q \cdot Var(x) \end{aligned} \tag{3.7}$$

Where equation 3.7 relates the expected value of the variance of the final trial vector to the variance of the initial population through L_q, a polynomial in q. Note that a value of

$L_q > 1$ implies that the variation of the trial population increases, whereas a value of $L_q < 1$ implies the opposite. We can find critical values of q by solving $L_q = 1$. We then obtain two roots: $q = 0$, or $q = 2 - 2F^2(NP)$. A detailed mathematical treatment of the parameter bounds discussing critical values are derived in Appendix B.

3.4 Results: Benchmark Set 1

Tests were conducted to compare the performance of DE, DE-SOM and GA with a suite of 15 typical benchmark functions for global optimization (see table A.1 in Appendix C). [73, 77, 89]. A wide variety of benchmark functions is required for testing any new algorithm. By this, the algorithm is prevented from taking advantage of specific features of any function, such as being symmetric, its optimum being at the center of the variable bounds, or optimum *at* the variable bounds. [90] Each algorithm was allotted five trials for optimization of each benchmark function. The best of five trials of each algorithm, DE, DE-SOM and GA were tabulated (see table 3.1). A small population size ($NP = 20$) is used to make convergence more difficult. The value of F and CR were fixed at 0.5 and 0.8 respectively, as recommended by authors in the past. [73, 77]

DE and DE-SOM were set to converge based on variance of the population, more specifically if $Var(x) < 1e-6$, whereas the *MATLAB* implementation of GA used here had five stopping criteria. [91] Our results show that this implementation of GA converges prematurely (see multiple entries of 1040 function evaluation). During several trials, GA was seen to 'get stuck' in local optima (see f13, for example). In all 15 cases, GA was outperformed by DE and DE-SOM (as seen in [73]). In terms of function evaluations, GA converged with lower number of function evaluations than DE for functions f3, f5, f7,f8,f9 and f15. However, as can be seen from the function value column, GA converged at a sub-optimal point for all these cases.

3.4.1 DE-SOM : Results of Two-Dimensional Benchmark Function Tests

As expected, DE-SOM performed better than DE with all 15 benchmark functions in terms of function evaluations. Table 3.1 only tests the functions in 2 dimensions ($n = 2$). The p value used in each trial may be adjudged by the ratio of function evaluations of DE to that of DE-SOM, and is adjusted during the multiple trials corresponding to one function (only the best of five trials is reported). When optimum function values are compared, we see that DE-SOM performs at least as well, if not better than DE in 10 of the 15 cases. For example, the row corresponding to the Easom function (f7 in table 3.1) shows that there is an error of $4e-7$ in the final function value obtained. However, we must acknowledge that the number of function evaluations saved is highly significant. For the Sphere function (f2 in table 3.1), the number of function evaluations of DE-SOM (320) is only a fraction (29%) of the total function evaluations taken by DE (1120). Benchmark functions f1, f4, f6, f10, f11, f12 and f14 are a few functions where large amounts of savings in terms of function evaluations are achieved. In these functions, the optimum value found is always as good or better than that of DE. Note that from the column on generations, that there is no exact algebraic relation between the number of generations and DE-SOM function evaluations as in DE. For instance, f12 and f14 converge after different number of generations, but have the same number of function calls(1040). In case of the Schwefel function (f13), DE-SOM converged to a value close to the reported optimum only once out the five trials corresponding to that row, whereas DE converged twice to a value close to -837, and GA did only found a local optimum.

In the case of DE or GA however, the number of function evaluations is always equal to $2 \cdot NP \cdot C$ (since each population member is used twice per generation). For functions f12, f13 and f14 the value of p was reduced to a very low value of 0.15. This means that the SOM learning is only used about 15% of the time. Nevertheless, function evaluations were reduced, and the quality of the final optimum value was not sacrificed. In case of the Schwefel function (f13), DE-SOM was seen to oscillate around a local optima in two of the five function trials. Reducing the value of p allowed DE-SOM to reach the known global

Table 3.1: Performance of DE, DE-SOM and GA across 15 benchmark functions

Function	Function Evaluations			Generations			Time		Function Value			f*
	DE	DE-SOM	GA	DE	DE-SOM	GA	DE	DE-SOM	DE	DE-SOM	GA	
f1	1120	240	1040	28	12	51	5	3.1	6.50E-03	1.00E-07	1.74E-02	0
f2	1120	320	1040	28	17	51	4.9	5.3	2.00E-06	1.00E-07	3.50E-03	0
f3	13440	2480	1040	336	66	51	50.1	11.6	2.00E-06	2.54E-04	1.60E-01	0
f4	1440	960	1180	36	28	58	6.136	5.396	3.001292	3.001496	3.4884	3
f5	1360	1200	1060	34	45	52	5.874	10.817	5.00E-06	8.00E-06	9.80E-03	0
f6	1320	160	1040	33	21	51	5.771	7.468	9.00E-06	1.00E-07	1.79E-02	0
f7	2600	1600	1100	65	48	54	10.341	12.347	-1.00E+00	-0.999996	-0.9974	-1
f8	2840	2560	1040	71	72	51	11.59	14.099	1.00E-07	1.00E-06	6.07E-02	0
f9	1720	1480	1160	43	50	57	7.252	11.339	5.50E-05	7.10E-05	5.06E-04	0
f10	2880	520	1060	72	24	52	12.542	6.88	1.00E-07	2.00E-06	1.93E-04	0
f11	4480	520	1040	112	44	51	18.629	13.622	2.00E-06	1.00E-07	1.18E-04	0
f12	1960	960	1040	49	37	51	9.461	10.297	6.46E-04	1.34E-04	9.39E-05	0
f13	2200	1880	3020	55	47	150	10.022	9.037	-837.356	-837.9657	-422.9234	-837.9568
f14	2120	1040	1040	53	31	51	9.648	7.455	1.00E-07	1.00E-07	7.64E-04	0
f15	1520	1240	1120	38	61	55	6.672	6.706	-1.8006	-1.8006	-0.6731	-1.8983

optimum. In fact, DE-SOM converged to a more optimal value than DE in this case. The column for comparison of time is only an indication of the fact that given simple functions, wherein an analytical function evaluation is almost instantaneous on any computer, nothing can be said about which algorithm is faster. However, given an optimization problem involving expensive engineering functions, the amount of time savings will also be as significant as the number of function evaluations saved. Since the functions are instantaneous, the time column represents time taken by the auxiliary functions (display, neural net training etc.), which is almost constant for a given population size. The time taken by expensive objective functions will render this time negligible.

3.4.2 DE-SOM : Higher Dimensional Function Results

Next, we performed tests on higher dimensional problems. 10-D and 30-D Versions of benchmark problems f1, f2, f3, f5, f10, f12 and f14 were used (see table 3.2). Again, each function was tested five times for a given dimension(10 or 30), and the best performance was recorded. Additional experiments with interesting results were recorded to show variation in performance due to change in control parameters. Of the functions listed in table 1, only some were extensible to larger than 2 dimensions. For all cases recorded in table 3.2, the population size was fixed ($NP = 30$). The dimension of the SOM network was also fixed, regardless of the dimension of the problem. This can be explained as follows. Consider an optimization problem that handles two-dimensional variables. Each trial vector contains 2 dimensions, and the function contour can be visualized in the third dimension. SOM neurons also have the same number of components as any population member (*two*, in the example point). However, the neurons can form a one-dimensional line network, a 2-dimensional area network or a 3-dimensional volume network for the purpose of moving towards and converging at the positions of extremum population members (see figure 3.3). In our second set of experiments, the SOM forms a sheet (as shown in the center of figure 3.3) in n-Dimensions ($kn = n - 1$).

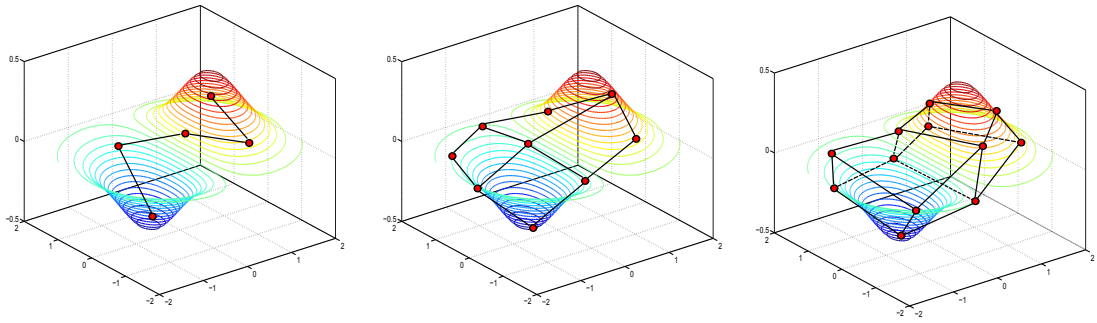


Figure 3.3.: Contour of a function involving variables with $n = 2$. The SOM network has neurons of the same dimension ($n = 2$), and may have links such that the SOM network formed is 1-D (left), 2-D (center) or 3-D (right).

In all cases, DE-SOM converged in lesser number of function evaluations when compared to DE. Another general observation across all test cases is that the variance of the final population was always lesser in the case of DE-SOM. For the benchmark function f1, the function value reached by DE and DE-SOM at convergence were sub-optimal. When DE is applied to f1 and f2, there is a sudden increase in number of function evaluations when transitioning from a 10- to 30-dimensional problem. On the other hand, the change in function evaluations is less obvious in the case of DE-SOM. The second 30-dimensional case for f2 and the second 10-dimensional case for f3 are worth mentioning since the number of function evaluations are much lesser than the cases used for comparison with DE, for a sub-optimal function value at convergence. The reader is reminded that these *extra* trials are reported since they are interesting, and are also one of the five trials corresponding to that benchmark function. The median function value for f3 was also reported to be sub-optimal by other authors using DE variants. [92] In the case of f5, the value of CR was decreased to 0.6 (from the default value of 0.8), and also, the number of replaceable elite population members was increased from increased from $NP/10$ to $NP/7$. Decreasing CR helped in accepting a larger percentage of mutant vectors, and the increased proportion of elite replacements helped escape local optima. DE-SOM converged after only a fraction of the function evaluations for the benchmark functions f10, f12 and f14. In these cases, the

Table 3.2: DE and DE-SOM performance comparison on higher dimensional ($n = 10$ and $n = 30$) benchmark functions

Function	Dimension (n)	Std. Dev		Function Value		f*	Function Eval	
		DE	DE-SOM	DE	DE-SOM		DE	DE-SOM
f1	10	9.68E-04	6.89E-04	0.00461	0.0048	0	5640	1120
	30	9.61E-04	8.99E-04	0.00436	0.00596	0	25440	3680
f2	10	9.18E-04	5.09E-04	0.00001	2.1E-05	0	5680	800
	30	9.55E-04	9.34E-04	4.8E-05	1.00E-07	0	25960	4240
	30	n.a	5.28E-04	n.a	3.2E-05	0	n.a	1200
f3	10	9.07E-04	9.06E-04	7.33223	7.7902	9	13280	7800
	10	n.a.	8.85E-04	n.a	9	9	n.a	440
	30	9.47E-04	2.65E-04	26.7968	26.7452	27	43640	1800
f5	10	4.99E-04	6.74E-04	5.2E-05	0.00019	0	15660	26520
	10	n.a	5.43E-04	n.a	2.97051	0	n.a	7520
	30	6.92E-04	6.69E-04	0.00145	1.96969	0	181740	147660
	30	n.a	5.81E-04	n.a	0.3017	0	n.a	68520
f10	10	9.05E-04	9.77E-04	1.9E-05	3E-06	0	19020	5580
	30	9.63E-04	8.66E-04	3.8E-05	1.7E-05	0	120180	4860
f12	10	9.64E-04	8.10E-04	0.00143	1.00E-07	0	93120	2360
	30	9.44E-04	8.75E-04	0.00217	0.00269	0	175680	3900
f14	10	9.40E-04	7.73E-04	2.1E-05	5E-06	0	19380	3720
	30	9.76E-04	8.61E-04	1.8E-05	2.8E-05	0	100860	3660

variance of the population was smaller for DE-SOM, and the function value was closer to the global optimum function value.

3.5 Airfoil Shape Optimization

This section reports the application of the DE-SOM algorithm to a benchmark airfoil optimization problem with the single objective of maximizing the efficiency factor, $\frac{Cl}{Cd}$. Although researchers have used different methods to parameterize the airfoil shape, their objective function has been $\frac{Cl}{Cd}$ (effectively). For example, Koziel *et al.* uses NACA pa-

rameters for lift-to-drag maximization [93]. Buckley and Zingg parameterize the airfoil geometry using B-spline control points, and perform lift constrained drag minimization [94] (as in Chernukhin and Zingg who use GA for the same, [95]). Our aim here is to expose the true utility of the DE-SOM algorithm by applying it to an optimization problem with a computationally expensive function call. Authors have used panel codes and CFD techniques to optimize airfoil shapes (for minimum drag, maximum lift, desired pressure distribution etc.). In our problem, the simultaneous maximization of lift and minimization of drag of an airfoil is captured by the single efficiency factor $\frac{Cl}{Cd}$. The first task is to choose an appropriate parameterization technique that has a large design space. Authors have used bezier curves, polynomials, splines and other curve fits to represent airfoils. [96–100] Here we use a recent method called Class/Shape Transformation (CST) introduced by Kulfan *et al.* for representing airfoil shapes. [101]. CST is chosen for our demonstration since it is able to represent a variety of airfoils using only six parameters (three parameters for each of the surfaces, upper and lower). [102] Unlike parameterization techniques that have specific parameters controlling specific features of the airfoil (for example, leading edge radius, camber etc.), the CST technique controls the shape of the airfoil indirectly. Therefore, our design variable x has six dimensions. The design space considered along with some sample airfoils is shown in figure 3.4.

The variable bounds were estimated by trial and error. The black-box function used to analyze airfoils a popular open-source tool *Xfoil*. In the trials conducted, a 'function evaluation' involves writing an input file to the *Xfoil* program, analyzing the airfoil that airfoil at a Reynolds Number (Re) of 10^6 and a Mach number (M) of 0.2 for a range of angle of attacks ($\alpha = -5$ to 25). The maximum $\frac{Cl}{Cd}$ (regardless of the corresponding α) is reported as the objective function value. In case *Xfoil* is not able to converge (usually due to an unconventional airfoil shape), we attribute that design point with a random, large, positive

number so that the same design point is avoided in the next generation. Our bounded, single objective optimization problem can thus be formulated in equation 6.25:

$$\begin{aligned} \text{Minimize} \quad & -\frac{Cl}{Cd} \\ \text{Subject To} \quad & LB \leq x \leq UB \\ \text{Where} \quad & LB = [0.05\ 0.05\ 0.05\ \text{-}1.00\ \text{-}1.00\ \text{-}1.00]^\top \\ & UB = [1.00\ 1.00\ 1.00\ \text{-}0.05\ 0.00\ 0.00]^\top \end{aligned} \tag{3.8}$$

Note that the problem is posed as a minimization problem with the appropriate sign change for the objective function. A graphical representation of the airfoil design space (which are formed by the six parameters in x) is shown in figure 3.4. A variety of airfoils with upper surfaces in the red region and lower surfaces in the blue region can be formed (a sample feasible airfoil is also shown). Again in this section, our experiment involves comprehensive tests to compare GA, DE and DE-SOM. Tests are performed to compare the three algorithms for cases with different population sizes (10, 20, 30 and 50). Each case (corresponding to a particular algorithm and a particular population size) is run thrice, and only the best case is reported (see table 3.3).

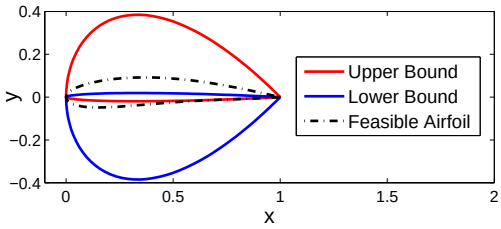
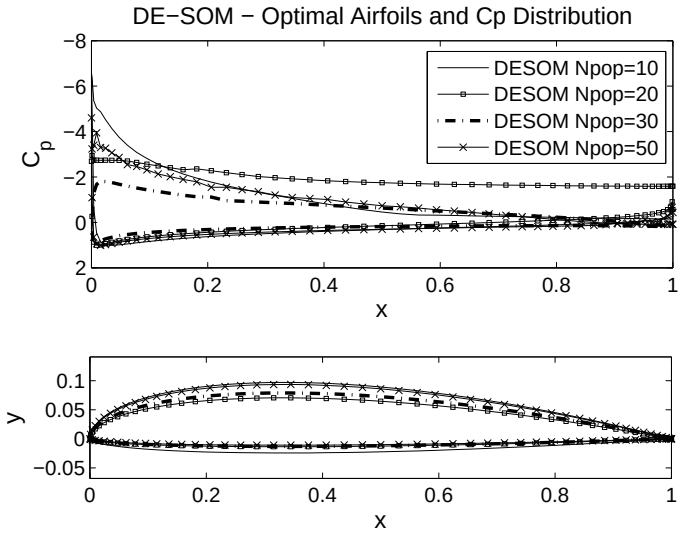


Figure 3.4.: Graphical representation of the airfoils formed with all six design variables at the lower bound (blue), and at upper bound (red). A feasible airfoil (one that lies between the upper and lower bounds) is also shown (black dashed lines)

We immediately observe the savings in number of function evaluations and time in the DE-SOM cases. For example, DE-SOM obtains values of $\frac{Cl}{Cd}$ close to those obtained by DE and GA (around 55), at a fraction of the function evaluations. The *Xfoil* black-box function used to evaluate each airfoil takes 5 to 10 seconds for each function evaluation. These benefits will be magnified for costlier black box functions. We also notice that GA with a population of 20 obtains a higher value of maximum $\frac{Cl}{Cd}$ than for any other case.



[h]

Figure 3.5.: Airfoils belonging to the different population cases that are results of the DE-SOM algorithm distinguished by their Cp distribution.

Table 3.3: Results of the airfoil optimization problem

Algorithm	**Npop**	**Generations**	**fun.evals**	**Time taken (s)**	**f***
GA	10	55	560	2.02E+03	-48.1578
	20	76	1540	1.48E+04	-60.8231
	30	51	1560	6.54E+04	-55.7278
	50	100	5050	5.81E+04	-56.6223
DE	10	57	1140	1.10E+04	-55.0891
	20	24	960	1.42E+04	-55.1958
	30	38	2280	2.22E+04	-55.4316
	50	43	4300	4.35E+04	-55.1854
DE-SOM	10	25	360	1.95E+03	-54.5445
	20	28	720	6.84E+03	-55.3173
	30	34	1260	1.27E+04	-55.1654
	50	45	3200	3.15E+04	-55.2247

However, this case takes almost as many function evaluations as the next case run with a population of 30. It should also be noted that there do exist other airfoils in this design space with a higher value for maximum $\frac{Cl}{Cd}$. For example, the sample airfoil shown in Figure 3.4 has a maximum $\frac{Cl}{Cd}$ that is greater than 200. Initial populations were random, and involved airfoils with large thickness, negative camber and even leading edge cusps. Figure 3.5 shows the airfoils that resulted from the DE-SOM runs for each population case. The corresponding coefficient of pressure (Cp) distribution is also shown to distinguish between the overlapping airfoils. Appendix D shows the final airfoils obtained for each of the 5 population cases for GA and DE. Larger population sizes and a larger mutation rate may be used for obtaining better designs.

3.6 Improvements made to produce DE-SOM2

The authors' experience with using DE-SOM has exposed the following shortcomings. *Firstly*, certain objective functions with multi-modal characteristics saw convergence to a local optimum instead of the global optimum. Note that this is an issue with several existing global optimizers. *Secondly*, the convergence rate depends on the parameters F, CR and p, the values of which are decided by the user. A situation that demands solving a new problem may not allow the user to fix these quantities using intuition or experience, especially since we are targeting costly black-box functions as our objectives. *Thirdly*, commercially available (or freely available) convex hull techniques cannot handle large dimensional problems ($> 9D$). SOM convergence is also sluggish for these large dimensional problems. This limits the use of our algorithm for a special class of large dimensional problems with expensive objective functions. *Lastly*, the SOM implementation involves saving function evaluations by using a continuously converging convex hull. The number of hits associated with neurons in the SOM network are only used in the elite replacement step, but may be used more effectively for building a virtual map of the function during the algorithm runs. These issues are discussed in the following sub-sections. Each of the suggestions are incorporated into DE-SOM2, an improved version of DE-SOM.

3.6.1 Premature convergence

From previous tests, the authors observed that some non-smooth, multi-modal objective functions, the algorithm converged to a local optimum. In these tests, the parameter p remained constant until the end of the simulation run. This implied that the probability of the algorithm chosen to be DE-SOM was finite until the end of the simulation. However, towards the end of the simulation, convergence ensures that the variance in the population members tends to a very low value (related to line of Algorithm in Appendix A). This poses a problem to available SOM implementations due tho the fact that several population members may be collinear, and this introduces bad scaling related issues. A subtle change introduced to fix this issue is the strict usage of DE at the final stages of the algorithm regardless of the p value. This change is made when the variance of the population is below $1e-1$ in DE-SOM2 (see line 4 of Algorithm in Appendix A).

3.6.2 Self-adaptive parameters

Several authors have chosen to use simple, self-adaptive strategies to change parameter values (like F and CR for DE) on the fly. [103–106] For evolutionary algorithms, self adapting involves targeting behavior of population members related to one of the following goals - exploration, exploitation, constraint handling or randomization. However, it is more common to base parameter values off a particular distribution (such as Cauchy or normal). Our self adaptation incorporates knowledge of performance of DE in the previous generations. Note that this is completely different from the more commonly seen choice of adapting the parameters according to DE performance based on a specific set of these parameters. Our view is that a given problem may be solved using several disparate choices for the set of control parameters (albeit with different convergence rates). Thus, it is more interesting to search for the right history of parameters, rather than the 'perfect' set of parameters that will solve the problem. We use self-adaptation for both F and p in our improved DE-SOM2 (lines 35-40 of Algorithm in Appendix A). For self-adapting the parameter p, we define new supporting, user-defined parameters $\gamma_{p1} > 1$ and $\gamma_{p2} > 1$, which represent the rates

of increase or decrease in the value of p. Obviously, p, which represents the probability of choosing DE over DE-SOM, ranges from $(0,1]$. Self adapting depends on monitoring the performance of the algorithm using a moving average window of fixed size. Let $favg_k$ represent the moving average function value (with a window size fixed to 10, or any other user specified value m) at the k^{th} iteration or generation. Now, $favg_{k+l}$ is compared with $favg_k$ to update the value of p (which is now a function of iteration number, therefore p_k). More precisely,

$$p_{k+l} = \begin{cases} min(p_k \times \gamma_{p1}, 1) & \text{if } favg_{k+l} \geq favg_k \\ max(p_k / \gamma_{p2}, 0) & \text{if } favg_{k+l} < favg_k \end{cases} \tag{3.9}$$

This promotes the informed choice of the parameter p during the algorithm run. Introduction of the γ_p parameters helps vary the rate of change of p. Similarly, we adapt the value of F across the interval $[F_{min}, F_{max}]$ if the performance becomes worse, or if the performance stagnates across a user specified window m. Notice that since F is a mutation parameter, we adapt the parameter randomly to encourage or discourage exploration (here *rand* represents a random number $\in [0,1]$):

$$F_{k+m} = \begin{cases} min(F_k / rand, F_{max}) & \text{if } favg_{k+l} \geq favg_k \\ max(F_k \times rand, F_{min}) & \text{if } favg_{k+l} < favg_k \end{cases} \tag{3.10}$$

3.6.3 Online surrogate modeling

SOM provides additional information about the function landscape that may be used to construct an online surrogate model that may further reduce the number of function evaluations. Recall that each neuron in the SOM network is associated with a certain number of population numbers (called hits). Now, given that we keep track of $f(x_i)$, the objective function values associated with the population members, and $hits_j$, the number of hits associated with the j^{th} neuron with wights w_j, we modify the potential method introduced by Takahama and Sakai to obtain the following *modified* potential method, which itself

performed well against benchmark functions when used in conjunction with DE. [107] Let $w_j \leftarrow x_i$ imply that the population member x_i is associated with the neuron with weight w_j. Notice that we could use the value of the mutant vector v_i, the trial vector u_i or the value of the population member x_i here. Let $d(x,w)$ represent the Euclidian distance between x and w.

$$\begin{aligned} &1.\ f(w_j) = \frac{\sum_i w_j \leftarrow x_i}{hits_j} \\ &2.\ U_o(x_i) = \frac{\sum_j hits_j \times f(w_j)}{d(x_i, w_j)} \\ &3.\ U_c(x_i) = \frac{\sum_j hits_j}{d(x_i, w_j)} \\ &4.\ \hat{f}(x_i) \approx \frac{U_o(x_i)}{U_c(x_i)} \end{aligned} \tag{3.11}$$

where the variables U_o and U_c are potential for the objective and potential for congestion respectively (as defined in the original potential method), and $\hat{f}$ is the required surrogate model of f (incorporated as a condition in line 24 of Algorithm in Appendix A).

3.6.4 Handling higher dimensional problems

As mentioned earlier, SOM implementations suffer from bad scalability for problems of higher dimensions. Solutions to this problem are usually in the form of batch or parallel implementations. [108] Since the size of the 'dataset' (here, the positions of all population members) is typically fixed to $50-200$, our problems arise in the next step - namely, finding the convex hull of the neuron position vectors w_j. Convex hull algorithms do not usually support triangulation in dimensions greater than $9D$. [87] This has implications in the choice of benchmark problems that can be used, as it is standard practice to test the new algorithm with problems of up to $100D$, let alone large scale optimization problems of up to $1000D$. Our workaround is to use a minimum volume covering ellipsoid (MVCE) for higher dimensional problems. We recommend using the Khachiyan algorithm to determine the MVCE, which is given as the solution to the following optimization problem [109]:

$$\begin{aligned} \text{Min.} \quad & \log(|A|) \\ \text{S. to} \quad & (x_i - c)^T \cdot A \cdot (x_i - c) \leq 1 \end{aligned}$$

with respect to the variables c and A, where c is the center of the ellipse in $\mathbb{R}^d$, and A is a $d \times d$ matrix of the ellipse equation in center form (i.e., $(x-c)^T A(x-c) = 1$). Here $|A|$ represents determinant of A, and x_i are the population members in $\mathbb{R}^d$. Note that even this workaround is a temporary one, since the calculation of the minimum volume covering ellipsoid involves a costly matrix inversion step. Other implementations of MVCE algorithms using interior point optimization techniques also exist in literature. [110]

3.7 Results: Benchmark Set 2

In this section, we use the lessons learned through benchmark tests in the previous section to test the suggestions incorporated into DE-SOM2. The problems considered here are the ones contained in the IEEE CEC 2005 benchmark set. [111] A compilation of the results by all participating algorithms are presented in Hansen [112], and a characterization of these functions as real-valued black-box landscapes over continuous domains can be found in Muller and Sbalzarini. [113] In this series of tests, we use $10-D$ variants of the compositional functions, which are treated as black-box functions. The success performance (SP) of an algorithm is used as a measure for the expected number of function evaluations (FE's) to reach a target function value (as defined by CEC 2005). The maximum number of function evaluations for all tests in $10D$ is 10^5. We run 25 runs on each of the 25 benchmark problems and compare the results obtained with other state-of-the-art algorithms as defined by Hansen. [112] Problem definitions are detailed in Suganthan *et al.*, but are repeated in the Appendix C for completeness. [114] A run is successful if the global optimum is reached with the given precision before 10^5 function evaluations is reached. We compare our algorithm DE-SOM2 with the 11 state-of-the-art algorithms compared in [112]. Note that in order to prevent exploitation of search space symmetry, problems presented are rotated, shifted and also hybridized with other functions. All convergence

graphs presented show the median performance of the total runs with termination criteria specified or due to exceeding 10^5 function evaluations. Performance measured on the plots show $\log_{10}(f(x)-f(x^*))$. Tests were conducted using MATLAB 2013a running on RHEL, powered by two 2.1 GHz 12 core AMD 6172 processors. For each function, we also report the best, median, and worst performance, along with mean and standard deviation of the runs measured at termination (described above). Two types of studies were conducted in the CEC 2005 competition - 1) Number of function evaluations to reach a fixed accuracy level (used in this study), and 2) Accuracy levels (best, median, worst etc.) at checkpoints of certain specific function values ($1e+03, 1e+04$ and $1e+05$). We compare DE-SOM2 with the performance of DE as reported in [115], [116] and [117]. Only two of these three authors who participated in the CEC 2005 benchmark competition with DE or DE variants report the average function values taken to attain a fixed accuracy ([115] and [117]). To keep our results consistent with Benchmark Set 1, we use the same conditions for convergence (variance of population $\leq 1e-6$). Note that for the practical problems we wish to solve using this algorithm, a variance-based convergence along with a function evaluation budget is more appropriate than a maximum function evaluation budget alone, since our scenario involves optimization under limited resources.

3.7.1 DE-SOM2 : Results for Unimodal functions

Results pertaining to functions 1-6 are reported in this section. Table 3.4 below summarizes the results obtained from 25 independent runs of these functions.

The table 3.4 shows the number of function evaluations, best value, median, worst and mean value, along with the standard deviation of the 25 runs at convergence. The convergence graphs shown in figure 3.6 shed more light about the convergence nature of DE-SOM2. For unimodal 10-D functions f1 to f6, we see good progress towards the optimum (plotted in terms of logarithm of the error). Functions f3 and f5 show poor or sluggish convergence (when considering median of all 25 runs), mainly due to the fact that the population is initialized randomly. For function f5, [114] states that if the population

Table 3.4: DE-SOM2 performance on 10 dimensional, unimodal CEC 2005 benchmark functions f1 - f6. The column Function Eval corresponds to the median number of function evaluations across all 25 runs.

Function	Function Eval	Best	Median	Worst	Mean	Std. Dev
f1	27900	2.371E-06	4.897E-06	2.891E+00	4.810E-01	7.266E+00
f2	58900	1.653E-06	1.083E+00	4.738E+02	1.895E+01	8.079E+01
f3	90100	7.438E+02	1.291E+03	3.432E+04	1.373E+03	6.864E+03
f4	58700	2.862E-06	8.906E-05	9.268E+00	9.497E-01	2.118E+00
f5	78400	4.261E+00	4.734E+01	1.171E+04	1.244E+02	2.450E+02
f6	91600	6.068E-01	5.557E-01	3.22E+00	8.910E-02	1.041E+00

is initialized on the bounds, "the problem may be solved easily". Clearly, f3 is the worst performing function among all unimodal functions f1 to f6. However, when compared to the parent algorithm's (DE) performance (see [117]), the reported best($1.34E+04$), median($1.47E+05$) and worst values($9.41E+05$) are lower for DE-SOM2 (compare with corresponding values in table 3.4). Similar comparisons can be made for f4 and f6. On the other hand, f5 converges to a lower best value of error than DE in lesser number of function evaluations, but has a higher worst value and similar median value of error. For other functions in which DE-SOM2 shows a high success rate such as f1 and f2, the global optimum is reached well before DE, which is also well before the maximum function evaluation budget is met.

3.7.2 DE-SOM2 : Results for Multimodal functions

The functions f7, f9, f10, f11, f12 and f15 are discussed in this section. As per Hansen *et al.*, these functions have been solved at least once by at least one algorithm in the CEC 2005 competition. The table 3.5 summarizes the results obtained from 25 independent runs of these six multimodal functions.

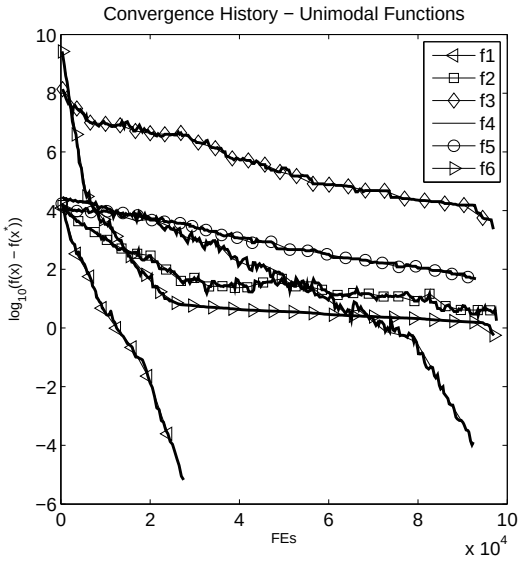


Figure 3.6.: Convergence histories for unimodal functions f1 to f6.

Table 3.5: DE-SOM2 performance on 10 dimensional, multimodal CEC 2005 benchmark functions f7, f9, f10, f11, f12 and f15. The column Function Eval corresponds to the median number of function evaluations across all 25 runs.

Function	Function Eval	Best	Median	Worst	Mean	Std. Dev
f7	34000	1.267E+03	1.2677E+03	1.267E+03	6.566E+02	6.484E+02
f9	98700	1.820E+01	1.820E+01	3.518E+01	1.441E+01	1.476E+01
f10	98700	2.572E+01	2.572E+01	4.011E+01	1.759E+01	1.753E+01
f11	99600	7.319E+00	9.210E+00	9.935E+00	9.110E+00	6.651E-01
f12	46500	9.596E-01	1.372E+01	7.123E+01	5.268E+01	1.476E+02
f15	76600	1.734E+02	1.437E+02	4.447E+02	1.671E+02	1.901E+02

As with other DE variants, performance for multi-modal functions is sluggish as seen in the convergence histories (figure 3.7). To comparing results with [115] and [117], we observe that the authors are not able to solve functions f7, f10, f11, and f12 (zero success rate). While [115] solved f9, [117] was able to solve f15. Due absence of final function values reported for zero success rate runs, a fair comparison could not be made with respect

to the multimodal functions based on progress of our algorithm. For example, although f9, f10 and f11 were not solved to the required level of accuracy, the error values are seen to gradually decrease towards the known optimum. We notice that DE-SOM2 performs very well with respect to f12, with which all three DE variants ([115], [116] and [117]) are seen to perform badly. Further investigation is required to ascertain why DE-SOM2 performs well in the case of f12 - a multimodal, rotated function which is asymmetrical and contains a large number of local optima.

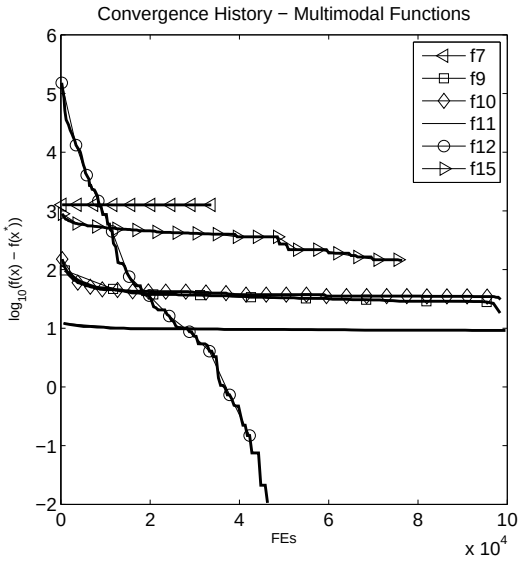


Figure 3.7.: Convergence histories for multimodal functions f7, f9, f10, f11, f12 and f15

3.7.3 DE-SOM2 : Results for Never solved functions

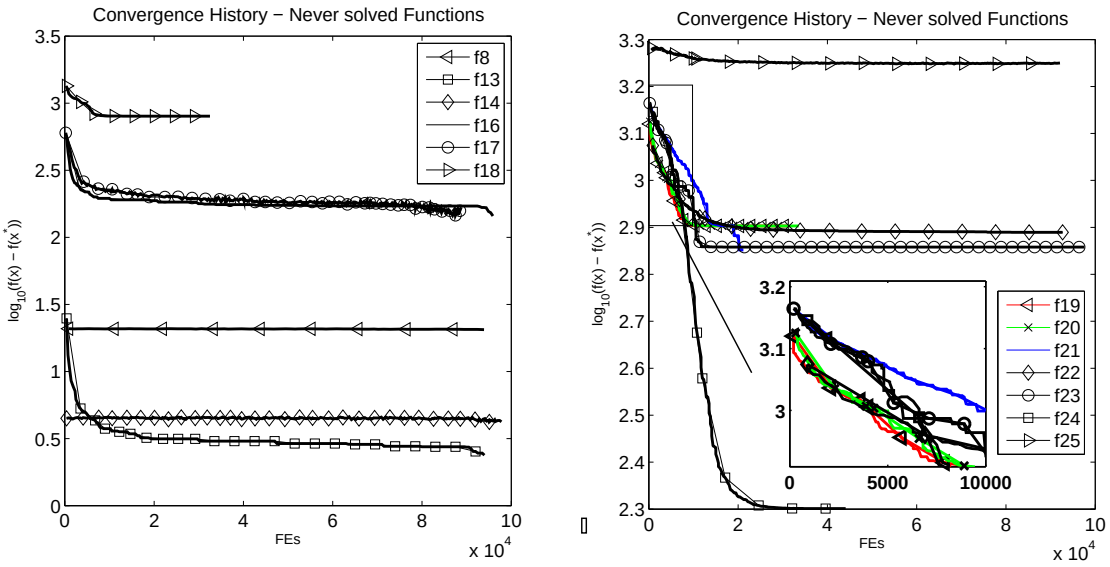
Results pertaining to functions f8, f13, f14, f16, f17, f18, f19, f20, f21, f22, f23, f24 and f25 are reported in this section. The table 3.6 summarizes the results obtained from 25 independent runs of these functions.

The convergence graphs corresponding to these never-before solved compositional functions are shown in figure 3.8 below. Our algorithm was not successful in finding the global optimum of any of these functions, similar to all other algorithms that competed in CEC

Table 3.6: DE-SOM2 performance on 10 dimensional, never-solved CEC 2005 benchmark functions f8, f13, f14, f16, f17, f18, f19, f20, f21, f22, f23, f24 and f25. The column Function Eval corresponds to the median number of function evaluations across all 25 runs. Note: More significant digits are added to the f24 row to distinguish between column entries

Function	Function Eval	Best	Median	Worst	Mean	Std. Dev
f8	86400	2.035E+01	2.065E+01	2.094E+01	2.064E+01	1.258E-01
f13	96600	1.822E+00	2.745E+00	3.518E+00	2.724E+00	4.578E-01
f14	89500	4.056E+00	4.562E+00	4.791E+00	4.515E+00	1.181E-01
f16	92800	1.441E+02	1.710E+02	1.879E+02	1.684E+02	1.262E+01
f17	86500	1.503E+02	1.784E+02	1.944E+02	1.740E+02	1.293E+01
f18	19500	3.000E+02	8.001E+02	8.009E+02	6.602E+02	2.291E+01
f19	21400	3.000E+02	8.000E+02	8.005E+02	6.601E+02	2.291E+01
f20	21200	3.000E+02	8.000E+02	9.637E+02	6.760E+02	2.248E+02
f21	15400	3.019E+02	8.007E+02	1.056E+03	7.156E+02	2.382E+02
f22	78900	3.000E+02	7.755E+02	7.854E+02	7.155E+02	1.579E+02
f23	98300	5.594E+02	7.121E+02	1.226E+03	7.201E+02	1.886E+02
f24	48600	2.000E+02	2.00003E+02	2.0001E+02	2.00004E+02	2.885E-03
f25	98400	1.692E+03	1.774E+03	1.786E+03	1.767E+03	2.333E+01

2005. Hansen *et al.* only compiles the rank of the median of the best function values obtained by all algorithms. When comparing final best values obtained by other DE variants in the competition, we notice that DE-SOM2 reaches a lower (or same) value for functions f8*, f13*, f14*, f16, f17*, f18*, f19*, f20, f21, f22, f23 and f24 (see [115], [116] and [117]. For starred(*) functions, DESOM2 performed better than at least one other DE variant). For functions where DE-SOM2 obtained the same value of error, lesser function evaluations were used.



(a) (b)

Figure 3.8.: Convergence histories for never-solved CEC 2005 functions f8, f13, f14, f16, f17, f18, f19, f20, f21, f22, f23, f24 and f25. The inset may be useful in differentiating between functions with similar convergence histories in that range (f19 - f22)

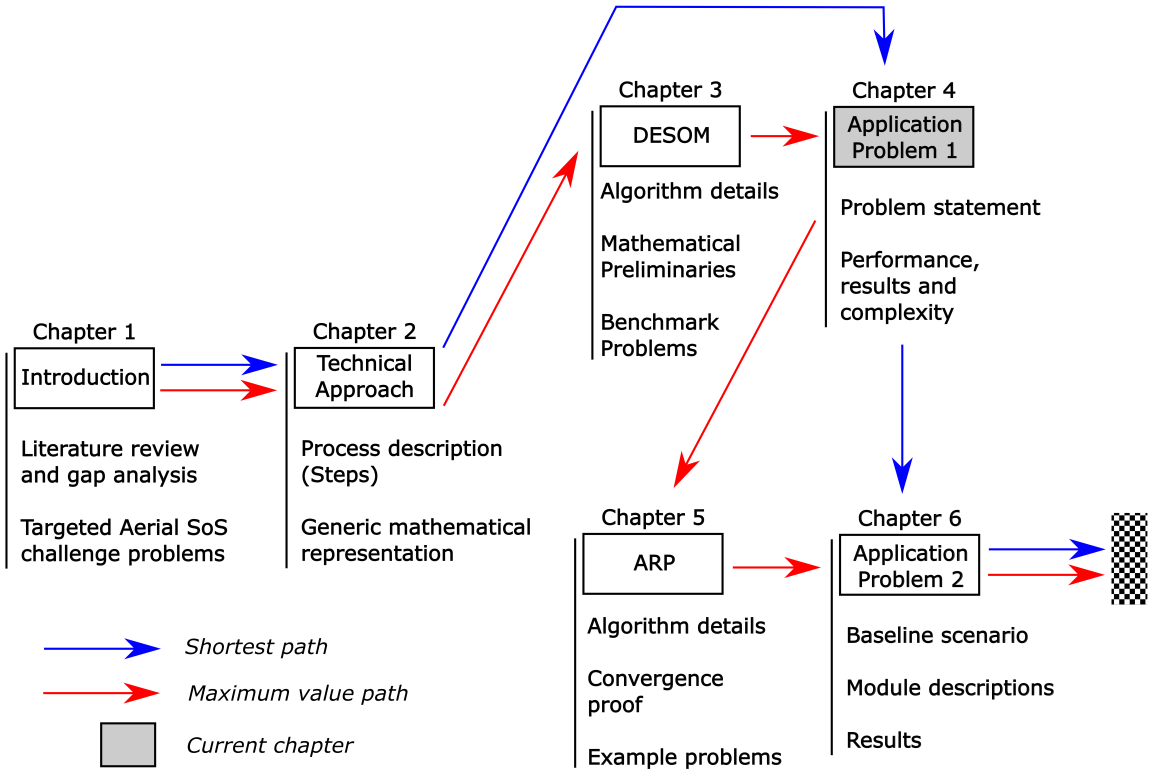
3.8 Section Conclusion

The performance of DE-SOM and DE-SOM2 were studied using two sets of benchmark functions. Using benchmark set 1, custom implementations of the DE and DE-SOM algorithm were compared to *MATLAB's* implementation of GA. Across 15 problems in benchmark set 1, DE-SOM performed better than DE (in terms of function evaluations and function value) and GA(in terms of function value). DE-SOM performed better than DE on higher dimensional benchmark functions by converging to the global optimum with less variability in the population and lower number of function evaluations. Functions in benchmark set 2 (IEEE CEC 2005) were solved using an improved version of DE-SOM (DE-SOM2), and compared with state-of-the-art DE implementations. DE-SOM2 converged to lower values of error with lesser function evaluations for several unimodal and multimodal functions. Of the 13 never solved functions in benchmark set 2, DE-SOM2

reaches a lower (or same) value for 12 functions. The effectiveness of the algorithm is also demonstrated in solving two real world application problems with objective functions that are computationally expensive (and that are not in the form of closed form equations).

The benchmark and application problems solved with the help of appropriate black-box functions successfully demonstrated that DE-SOM and DE-SOM2 find high performance designs as compared to DE and GA. While DE-SOM converges to better designs than DE after fewer function evaluations, DE-SOM2 improves the design further using additional function evaluations. Our SOM implementation is generic, as this can be applied to any other evolutionary algorithm that is set up in the typical format representing the steps of an evolutionary algorithm, namely *mutation-crossover-selection*. The improvement to be expected while using SOM as an add-on is convergence to a higher or similar quality answer as the parent algorithm, but with lesser function evaluations. Extension of the algorithm to constrained and stochastic optimization remains a challenge for our future work. It is important to improve the SOM implementation to handle constraints, bounds and multiple objectives directly, and in a computationally effective manner.

We have now described tools required to solve the precision agriculture UAV swarm problem. Other tools required to solve the second application problem will be described in the following chapters.



We now proceed to describe and solve our first challenge problem. Although the scale of the problem in terms of number of computational workers used, time taken to obtain a solution at all three levels of the SoS at hand, and amount of detail/ fidelity involved is small, it involves unique features such as evolving design spaces through numerical continuation and a suitable computational architecture (Single Program Multiple Data). Since we attempt to solve this problem on a time budget and a shared set of processors (1 worker), monitoring performance and complexity of the process and the product at each level using the hierarchical complexity metric becomes more relevant (than in application problem 2).

4. Application Problem 1

4.0.1 Problem Description

An SoS consisting of one or more aircraft is to be used for surveying a plot of land of width $S_x = 20$ km and height $S_y = 10$ km. Each aircraft is equipped with an imager that can resolve a strip of 10 m width at cruising altitude. note that although the cruising altitude changes for different aircraft, the imaging resolution remains the same by changing the sensors themselves. At the SoS level, the aircraft chosen to form the SoS are simulated over the plot of land described. The wind conditions over the plot of land at the cruising altitude vary randomly (uniform random about mean zero), and hence the trajectory of the aircraft simulated varies from an ideal one. Each aircraft is assumed to travel at a constant speed (its designated cruise speed). The minimum radius of curvature (corresponding to the maximum G-loading) varies with respect to the cruise velocity of the vehicle. At the SoS level, the design variables are :

1. Aircraft Name / index - this allows us to access the designated cruise velocity (Varies from 1 to number of aircraft at that point of time in the library, for each aircraft considered in the SoS)

2. Final weight fraction to calculate theoretical endurance. (Varies from 0.5 to 0.9 - theoretical maximum and minimum for demonstration purposes.)

3. Specific Fuel Consumption (SFC) of the engine used. (Varies from 0.1 lb/lbf-hr to 3.0 lb/lbf-hr).

An aircraft is simulated along a typical “Lawn Mower Surveillance Pattern” and continues on this pattern until the edge of the plot is reached, or the fuel remaining is just enough for the aircraft to return home (see figure 4.1. Note that the aircraft’s imaged area

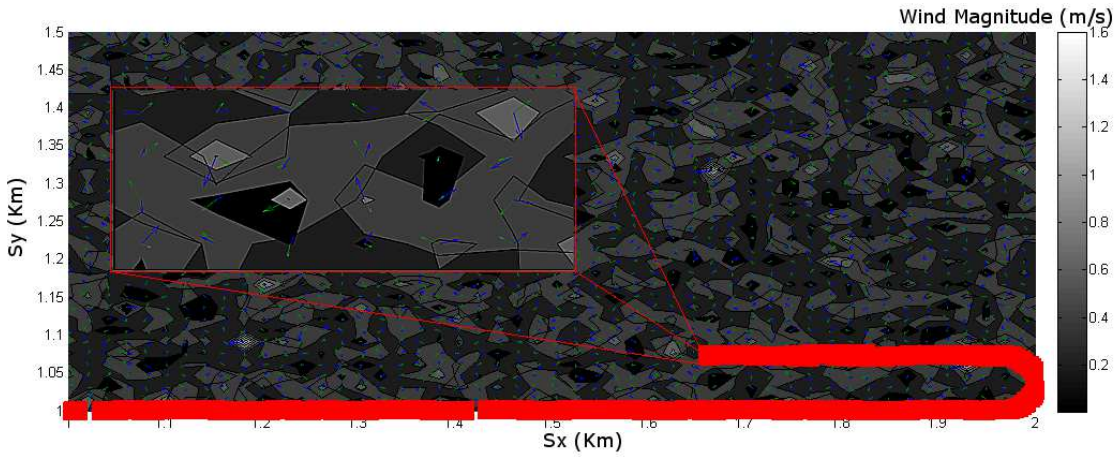


Figure 4.1.: Part of the typical lawnmower path adopted by aircraft at the SoS level. The inset shows a magnified view of the wind contour, colored by magnitude in m/s. The small blue and green arrows indicate the (random) initial and final positions of the wind vector during the SoS level simulation

depends on the distance travelled and the image resolution on the ground. We assume that in the presence of multiple aircraft in an SoS, the mission is suitably varied so that there is not much overlap between the areas captured by the multiple aircraft. Thus, when an aircraft returns to the home location for lack of fuel or otherwise, the next aircraft begins imaging the plot from last position imaged by the previous aircraft. For the purpose of this demonstration, all aircraft start from the home position. We require to find a set of aircraft that when combined, image most of the plot of 20 km by 10 km.

Ideally we would want an aircraft that covers the entire plot while exactly using up the fuel available. Suppose p is the percentage of endurance of the vehicle used. The excess endurance is thus given by $p_e = 100 - p$. Let us define the Area Ratio $A = S_x \times S_y/(\text{Area Imaged})$. Thus, an optimal SoS would minimize the sum of the excess endurance p_e and the Area Ratio A. The number of aircraft in the SoS is evolved from one to a maximum of five, since it is expected that less than five aircraft will be required to survey a plot of land of the given size. A custom Agent Based Model (ABM) is used as the objective function.

The SoS selects aircraft from the System level library to form potential SoS. The aircraft themselves are part of a library or database of aircraft, which is continuously updated to include new *optimal* aircraft that the SoS can select. The design variables at the system (aircraft) level describe the wing of an aircraft.

1. Root Chord
2. Taper Ratio
3. Sweep
4. Dihedral
5. Span
6. Root airfoil
7. Tip airfoil

An open source Vortex Lattice Method code, *Tornado*, is used to simulate the aircraft across several operating points for the appropriate cruise velocity and angle of attack for the best endurance factor $C_L^{\frac{3}{2}}/C_D$, which is the objective to be maximized at the system level. Although the design variables form a simple wing with two sections, the other aircraft in the library are detailed and involve geometry such as tails, control surfaces and bodies such as fuselages and engines. Unlike the SoS library that is initially empty, the aircraft library has 32 aircraft before the beginning of the simulation. New aircraft are added as the aircraft optimization problem progresses. An example of an aircraft from the library is shown in figure 4.2 below:

The airfoil optimization problem used in 3.5 is repeated at the Sub-system level. Readers may re-visit that section if required. The airfoil, although represented by the CST parameterization for the purpose of optimization, is interpreted as a number of panels by the Xfoil black-box program. Increasing the number of panels increases the fidelity of the solution. Note that the VoI metric is not used in this case for simplicity. In our demonstration, we gradually increase the number of panels from 50 to 150 in multiples of 10. The reader is reminded that the *design space converges* if say, the optimum airfoil with

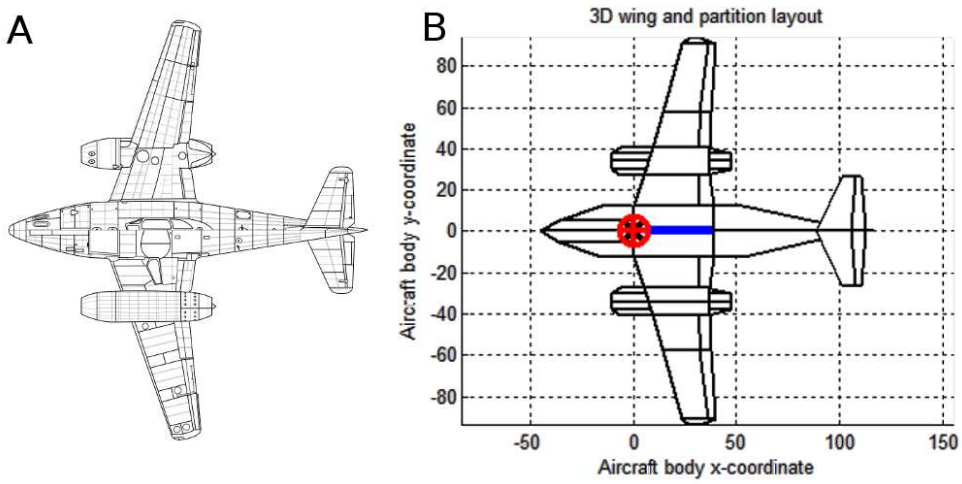


Figure 4.2.: A: A top view drawing of the Messerschmitt Me 262, an aircraft in the library of aircraft, and B: Representation of Me 262 in the VLM code Tornado

50 panels is almost the same as that with 60 or 70 panels on its surface. The final SoS optimization problem can be represented as in table 4.1

A summary of the algorithms used at each level and other relevant details are given in table 4.2. Note that all problems were bound-constrained, and other implicit constraints (for example aircraft 2D dynamics in the ABM or template for forming new aircraft) are hard-coded in the actual implementation.

In the context of the current problem, the framework we introduced in Chapter 2 is depicted in Appendix A. The multi-level and multi-fidelity aspects are clearly depicted, and the indicated steps involved are described in detail in this section. Foundational elements of this framework such as Platform-based Design (PBD) paradigm, evolving design spaces, multi-fidelity analysis and hierarchical complexity are also described as part of the process description.

4.0.2 Complexity Assessment

Figure 4.3 shows the function control graph of the program used to implement the above application problem. The graph is essential for the calculation of Cyclomatic Complexity

Level	Relevant Equations	Comments
Iterate	$G = G+1$ if $\begin{cases} (C_v \leq C_v^{max}) \wedge (C_h \leq C_h^{max}) \text{ and} \\ P^{predicted} \geq P^{min} \end{cases}$	Move to next Generation if thresholds satisfied
Converge	$X_{sos}^{lib,G+1} = X_{sos}^{lib,G}$ if $\begin{cases} G > G^{max} \text{ or} \\ f_{sos}^{G}(X_{sos}^{*}) \rightarrow f_{sos}^{G+1}(X_{sos}^{*}) \text{ or} \\ f_{sos}^{G}(X_{sos}^{*}) \leq f_{sos}^{G+1}(X_{sos}^{*}) \end{cases}$	Conditions to stop evolving component libraries
SoS	$X_{sos}^{lib} = X_{sos}^{lib} \cup \{X_{sos}^{*}\} \vert \text{Opt.} f_{sos}(X_{sos}) \vee g_{sos}(X_{sos}) \leq 0$ $X_{sos} = \{X_{sys}^{k}, X_{sys}^{k+1}, \ldots, X_{sys}^{k+l-1}\}$ $l \leq \vert X_{sys}^{lib} \vert$	Assemble SoS from systems. Append Optimal/feasible SoS components (X_{sos}^{*})
System	**Subject to** $X_{sys}^{lib} = \text{Opt.} f_{sys}(X_{sys}) \vee g_{sys}(X_{sys}) \leq 0$ $X_{sys} = \{X_{sub}^{k}, X_{sub}^{k+1}, \ldots, X_{sub}^{k+m-1}\}$ $m \leq \vert X_{sub}^{lib} \vert$	Form optimal systems (X_{sys}^{*}) from evolving sub-system library. Note that this level itself does not evolve.
Sub-system	**Subject to** $X_{sub}^{lib} = X_{sub}^{lib} \cup \{X_{sub}^{*}\} \vert \text{Opt.} f_{sub}(X_{sub}) \vee g_{sub}(X_{sub}) \leq 0$ $X_{sub} = \{X_{cmp}^{k}, X_{cmp}^{k+1}, \ldots, X_{cmp}^{k+n-1}\}$ $n = \vert X_{cmp}^{lib} \vert$ Where $X_{cmp}^{lib} = \{X_{cmp}^{1}, X_{cmp}^{2}, \ldots, X_{cmp}^{n}\}$	Form optimal sub-systems (X_{sub}^{*}) from fixed library of airfoils

Table 4.1: Summary of the SoS optimization problem with level-specific equations

(C_v) which is one of the parameters used to control the overall process. As we can see from table 4.3, C_v is maximum for the lowest level (airfoil optimization), and minimum for

Level	**Optimization / Algorithm**	**Details**
SoS	**GA** Min $A_{total}/A + Endu._{excess}$	•Library initially empty •3 variables per aircraft node (Id., Weight fraction, SFC) •Objective function is a custom ABM
System	**GA** Max. $C_L^{3/2}/C_D$	•32 aircraft in library •7 variables (Chord, Taper ratio, Sweep, Dihedral, Span, Root and Tip airfoils) •Option to create new aircraft based on template •Tornado VLM + Pablo
Sub-System	**DESOM** Max. C_l/C_d	•88 airfoils in library •6 variables •Adjust fidelity through number of panels •Xfoil (viscous)

Table 4.2: Details of optimization problem formulated in each level

the SoS level. Note that the SoS, System and Sub-system level programs run in parallel, and interact only through the common libraries (marked 'Airfoils' and 'Aircraft'). For measuring Process Performance (P_v), we notice from the same control graph that the SoS program has a low fidelity end state, whereas the System and Sub-system programs have one high fidelity end state each (represented as black-boxes 'Xfoil' and 'Tornado'. Process Performance is high if a particular level finishes jobs involving higher fidelity processes in quicker manner. Table 4.4 shows Process Performance (P_v) as a function of time. Thus C_v and P_v are used to either manually or automatically control the process.

Product complexity (C_h), on the other hand, is controlled by the SoS level. In our application problem, we assume that the aircraft are fully connected, and the nodes are the

Level	**No. of Edges (E)**	**No. of Nodes (N)**	**No. of end states (P)**	**Cyclomatic Complexity ($C_v = E - N + 2P$)**
SoS	6	6	1	2
System	7	6	2	5
Sub-system	8	8	4	8

Table 4.3: Calculation of Cyclomatic Complexity of programs used in optimization of each of the three levels

Level	**No. of Fidelity 1 Processes**	**No. of Fidelity 2 Processes**	**Process Performance $P_v(t)$**
SoS	6	0	$6/t$
System	6	1	$8/t$
Sub-system	7	1	$9/t$

Table 4.4: Calculation of Process Performance of programs used in optimization of each of the three levels as a function of time taken for completing the process (t)

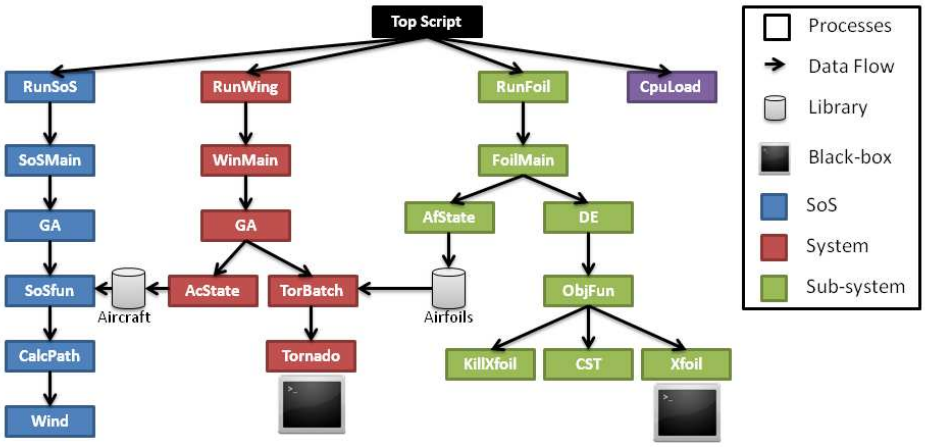


Figure 4.3.: Function control graph of the program used to implement the SoS optimization framework for the current application problem. The graph is used to calculate Cyclomatic Complexity, C_v

aircraft themselves. Thus for n aircraft, C_h is equal to the sum of number of components and number of links, or $C_h = n + \frac{n\times(n-1)}{2}$. Since optimization is performed in each level to maximize product performance, the fourth characteristic graph is irrelevant.

Target specification is also incorporated through components in the library. For example, optimal airfoils are added to the library at regular intervals (checkpoints in the optimization process) and may or may not be used by aircraft in the system level unless an improvement in the objective function of the lower level is seen. Optimal aircraft that continuously do not make use of the new optimal airfoils being added to the sub-system library is an indication of low utility of these airfoils across levels. These airfoils are therefore removed from the library. Copies of sub-optimal or optimal airfoils that are used by optimal aircraft are placed in the airfoil library to increase the chance of selecting these airfoils for system-level tests. Similar addition and removal of components is carried out in the system level aircraft library.

4.1 Results

The three level ('SoS', 'Sys' and 'Sub') SoS optimization problem was implemented in *MATLAB*'s parallel computing toolbox using SPMD (Single Process Multiple Data). The maximum allowable time was set to 500 minutes (or 30000 seconds, as seen in the upcoming plots). The three 'streams' of code corresponding to each level were started synchronously. As mentioned earlier, the SoS and sub-system levels are evolving levels, whereas the system level is static. The reader is reminded that aircraft are continuously added to the system library, but the number of variables used to describe the aircraft is constant (7). The SoS level grows from having to select 1 aircraft to 5, and at the sub-system level, the number of panels on an airfoil is changed from 50 to 150. One of the main thrusts of incorporating hierarchical complexity is to manage computational intractability. However, in this example we use a single fidelity of tools at the end state of each function. In other words, the objective function exists at a single fidelity level; either low fidelity (fidelity= 1) or high fidelity (fidelity= 2). No level offers the overseeing hierarchical complexity manager a choice of fidelity. Nevertheless, our demonstration clearly suggests where, how and when these 'choices' can be made.

During the implementation of this problem, two new airfoils and one new aircraft were added to the corresponding level-specific libraries. Optimization at each level limits the following:

1. Product Performance - At each level, the performance at each level is maximized (appropriately interpreted as maximizing and minimizing the objective function subject to constraints and variable bounds).

2. Product Complexity - The maximum number of a) panels on an airfoil (150), b) variables to describe an aircraft (7) and c) nodes that form an SoS are fixed.

Thus, as part of our results in this section, we will only be visualizing the Process performance and Process Complexity. Results are described in terms of five important milestones during the progress of the simulation (see table 4.5).

Milestone	Time (s)	Significance
1	1879	SoS evolves Once (Uses aircraft generated from System level for optimum SoS)
2	5594	SoS evolves again (Uses Me 262 and B47 B, rejects older optimum)
3	11016	SoS design space converges (Function value increases. Uses three other aircraft at optimum)
4	27391	Airfoil optimization ends
5	30000	Aircraft Optimization is stopped

Table 4.5: Milestones during the simulation along with their significance (To be used for interpretation in subsequent tables)

Figure 4.5 shows the time history of the total process complexity and process performance throughout the simulation period. Initially, both total performance (Pv) and complexity (Cv) decrease as expected. Complexity Cv only decreases when the SoS design space converges (third milestone). Until that point, all three levels are active. Performance decreases since the SoS level is growing in size (evolving) during the first two milestones. It must be noted that at the end of the first epoch of evolution, the SoS level uses the aircraft 'Simple2' that is generated by the system level in its optimum result. The aircraft 'Simple2' is however, not an optimum aircraft with respect to the system level. Nevertheless, we add another instance of this aircraft to the system library since it is useful to the SoS level. After the second epoch of evolution in the SoS level, the aircraft used are the Messerschmitt 262 and the Boeing B-47 Stratojet. The path of taken by these aircraft with the optimum parameters is shown in figure 4.4. Note that the second aircraft (here, the B-47) starts imaging from the spot that the first aircraft (Me-262) stopped. Also, there is no excess endurance, since both aircraft have spent the entire amount of fuel, and have imaged most of the area. Although uncertain wind was used in these simulations, wind contours are not shown in figure 4.4 for clarity. The plot is to be interpreted as areas which

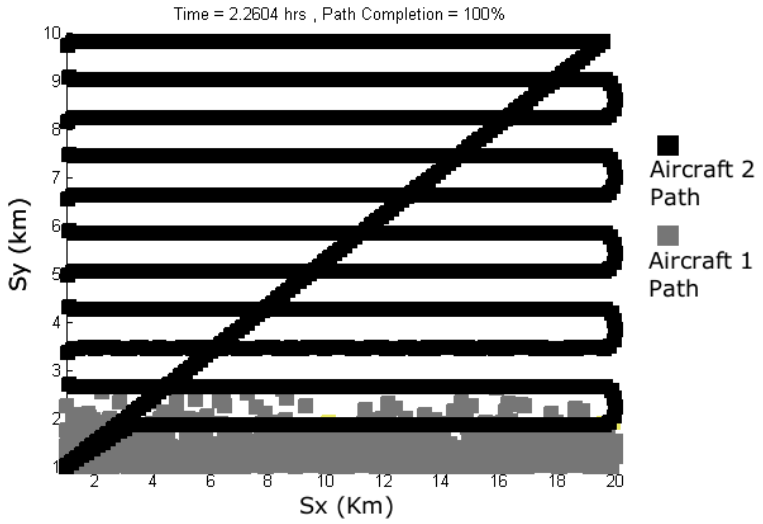


Figure 4.4.: Path taken by the optimal SoS of aircraft to complete surveillance over the 20 km by 10 km plot of land.

have been imaged (denoted by color-coded squares) below the trajectory taken by each aircraft in the SoS. Note that in our application problem, the SoS level is the focus. Other application problems may consider the system level to be the focus, for example, and may not even have an SoS objective function (according to this framework, it must at least have constraints or variable bounds).

The SoS design space is said to have 'converged' at milestone 3 since the optimal objective function value is worse than the optimum value after the second evolution. After this point, the SoS code is stagnant, and may either wait for completion of other levels or may be re-run to validate results. We further study the impact of these processes in terms of performance and complexity using the figures 4.6 and 4.7. In figure 4.6, we see that the performance decrease is dominated by the fact that the SoS level stops operating after the third milestone. Thus after milestone 3, computationally heavier Sub and Sys levels are allocated all the available resources. As time progresses, the convergence of the airfoil optimization code using *Xfoil* causes all computational resources to be allocated to the System level, which is stopped prematurely due to the time constraint. Since cyclomatic complexity is not dependent on the time between each milestone, the stacked bar graphs

seem more regular. The number of levels seen at a particular milestone represents the number of active levels at that point of time. As we can see, the optimization problems at the SoS and Sub levels have converged at least once by milestone 4, and the aircraft optimization is active throughout the simulation time.

Thus we are able to make the following conclusions for the benefit of those who would like to replicate or modify these results in the future:

1. To actively control the performance and complexity of the entire simulation, lower fidelity end states must be added. Here the 'black box' applications *Tornado* and *Xfoil* are to be complemented by lower level, heuristics based or vote based objective functions.

2. The current results (Pv and Cv) may be used as *orders* or thresholds for any following analyses where Hc and Hv can be calculated continuously to make level specific decisions.

3. Evolutionary algorithms are used for optimization at each level. Instead, analytical functions may be developed for use with standard gradient based algorithms for initial design space exploration, whereas more flexible evolutionary algorithms can be used for final convergence.

4. Since it was expected that the most computationally intensive level would be the system level, a non-evolving aircraft optimization problem was chosen. Adding multiple fidelity end states at the system level may relax this constraint and allow for an evolving design space.

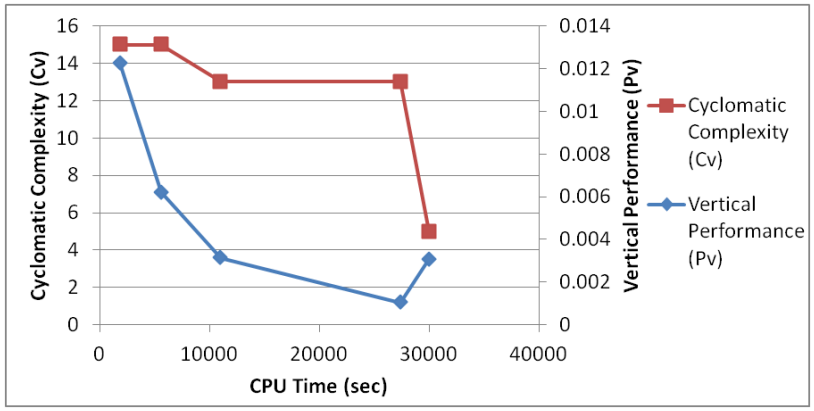


Figure 4.5.: Variation of total (all three levels) *Cv* and *Pv* with time

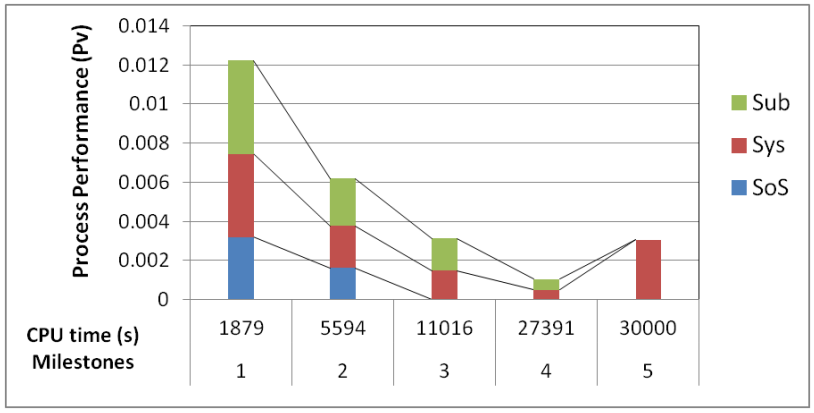


Figure 4.6.: Vertical performance variation with respect to time along with contribution of each level

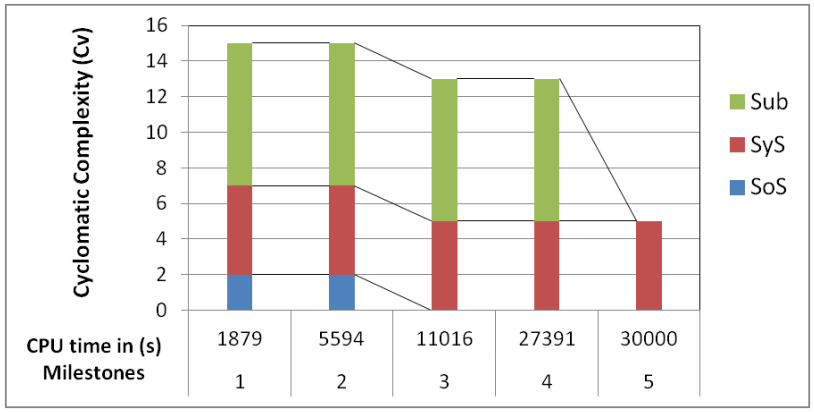
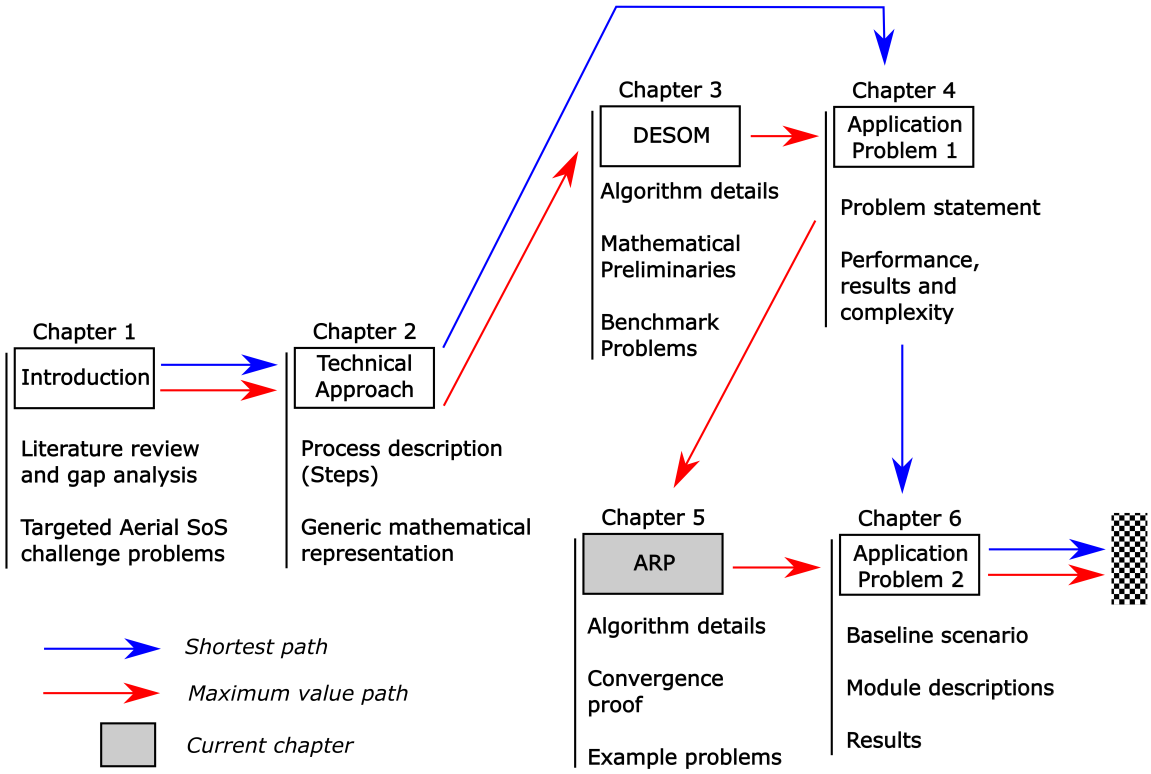


Figure 4.7.: Cyclomatic complexity variation with respect to time along with contribution of each level



The vastly unexplored area of optimization under evolving design spaces are dealt with in this chapter. We provide a provably convergent algorithm for solving a special class of problems involving change in dimension, and demonstrate the use of this algorithm (adaptive Random Projection) on real-world application problems.

5. Dual averaging with Adaptive Random Projection (ARP) for solving evolving distributed optimization problems

Before we present our final application problem, we discuss a second type of evolving design space [1], and its corresponding solution method. In this chapter, we study a sequential form of the distributed dual averaging algorithm which minimizes the sum of convex functions in a special case where the number of functions increases gradually. [2] This is done by introducing an intermediate 'pivot' stage that is posed as a convex feasibility problem that minimizes average constraint violation to a family of convex sets. By doing so, we introduce a version of the minimum sum optimization problem that incorporates an evolving design space. Two very relevant, and popular problems are solved using our method in the setting of evolving design spaces - finding a robust source location in a wireless sensor network, and minimizing the compliance of a structural topology domain. Results obtained confirm that the new designs in the evolved design space are superior due to the unique path followed to reach the optimum.

5.1 Introduction

We are interested in convex optimization problems of the form shown in equation 5.1. Define problem at time t, $\mathscr{P}_t$ as follows

$$\mathscr{P}_t : \begin{array}{l} \min\limits_{x} \sum\limits_{i=1}^{n} f_i(x) \\ x \in X \end{array} \tag{5.1}$$

[1] Type one was discussed earlier, and was solved using numerical continuation

[2] This special form of the objective function (sum of individual convex functions) is used commonly in distributed optimization applications.

where $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function (may not be differentiable), and X is a non-empty, closed and convex set with $X \subseteq \mathbb{R}^n$. In our case, the optimization problem at time t, $\mathscr{P}_t$ may evolve at a certain time (say $t+1$) to include p more individual functions, for example problem $\mathscr{P}_{t+1}$ may be given as:

$$\mathscr{P}_{t+1} : \begin{array}{c} \min\limits_{x} \sum\limits_{i=1}^{n+p} f_i(x) \\ x \in X \end{array} \tag{5.2}$$

with $X \in \mathbb{R}^{n+p}$. To simplify notation, we can consider an equivalent optimization problem in equation 5.3 which is based on functions that are distributed over a network specified by an undirected graph at time t, $G_t = (V_t, E_t)$ over the vertex set $V_t \subseteq \{1,2,...,n\}$ with edge set $E_t \subseteq V_t \times V_t$.

$$\begin{array}{c} \min\limits_{x} \sum\limits_{i \in V_t} f_i(x) \\ x \in X \end{array} \tag{5.3}$$

To relate equations 5.2 and 5.3, we note that adding a vertex to the network is akin to adding a set of p variables to the optimization problem. Each vertex is associated with an agent that calculates a local function f_i, typically a function of p variable dimensions of the total n dimensions in $x \in X, X \subseteq \mathbb{R}^n$.

Define $f := \sum_{i \in V_t} f_i(x) \mid x \in X$. Let $f_t^* = f(x) \mid x_t^* \in X$ solve problem $\mathscr{P}_t$. Furthermore, define $t+ := \{t' | t' > t\}$. We discuss our answers to the following questions through this paper:

1. Given an optimization problem ($\mathscr{P}_0$) of the form in equation 5.3 that is associated with a graph G_0 at time $t = 0$, how do we sequentially obtain better f_{t+}^* as new vertices are included in the graph G_{t+}? This also defines our motivation - to find a minimum such that $f_{t+}^* < f_t^*$.

2. Is there any benefit of finding the trace of f_t^* sequentially rather than solving a static problem at t_{end}?

3. When does one stop considering the option of changing the number of vertices ($n(V_t)$) in graph G_t to obtain a final $f^*_{t_{end}}$?

5.2 Foundations of the algorithm

Problems of the form shown in equation 5.1 have received much attention in the recent years due to their applicability to a variety of fields and disciplines. Several branches of optimization problems and their resulting applications are focused on optimizing 'additive cost' as in this paper. Some examples include Least squares and Inference, Stochastic Programming, Dual optimization and Machine Learning. [118, 119] Bertsekas provides a unified framework, along with convergence and convergence rates of a variety of methods that can be derived from three basic solution strategies - incremental gradient, incremental subgradient and proximal methods. [119] However, all of the methods and variants presented in literature only tackle a problem of fixed size (n). Although Nedic and Olshevsky [120] use a subgradient method for solving a distributed optimization problem over time varying graphs, the number of vertices remain fixed (i.e. V does not vary with time). We present a novel method that combines Distributed Dual Averaging (Sect. 5.2.1) and the Random Bregman Projection method (Sect 5.2.2) that solves our version of the time varying minimum sum problem as seen in equation 5.3.

5.2.1 Distributed dual averaging

From the perspective of decentralized or distributed optimization, the goal is to optimize a 'global' objective by only 'local' (or node specific) function evaluations. Practical implementation of these optimization algorithms to tracking and localization problems, sensor networks and multi-agent coordination have proved to be very effective in the past. [121, 122] Although the form of the function handled by Duchi *et al.* [123] is slightly different, our work is an offshoot of the method presented therein, and is re-purposed to effectively handle an evolving design space along with the techniques in Section 5.2.2.

Basic assumptions

Standard dual averaging schemes are based on a proximal function $\phi : X \to \mathbb{R}$ that is strongly convex with respect to some norm $\|\cdot\|$. In our paper, we use the canonical form of this proximal function, a quadratic of the form $\phi(x) = 1/2\|x\|_2^2$ which is strongly convex with respect to the L_2 norm for $x \in \mathbb{R}^n$. Also, we assume that all nodal cost functions (f_i's) are $L-$Lipchitz with respect to each norm as in Duchi *et al.* [123] That is,

$$|f_i(x) - f_i(y)| \leq L\|x-y\|, \forall x, y \in X \tag{5.4}$$

The functions used in in this chapter are all convex functions in compact domains, and hence satisfy this condition. Note that a formal proof is possible only using this assumption, but our demonstrations in forthcoming sections are not limited to convex problems. An important inference of equation 5.4 that is used in the proof of convergence of the Distributed Dual Averaging (DDA) algorithm, is that for any $x \in X$ and any subgradient $g_i \in \partial f_i(x)$, $\|g\|_* \leq L$.[3]

Define the adjacency matrix $A_t \in \mathbb{R}^{n\times n}$ of graph G_t as follows:

$$A_t(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E_t \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

Each node in the network may be associated with neighbors $j \in N(i) := \{j \in V_t \mid (i,j) \in E_t\}$. Let degree of each node be defined as $\delta_i := |N(i)| = \sum_{j=1}^{n} A_t(i,j)$, and $D = diag\{\delta_1, \ldots, \delta_n\}$. Let $\delta_{max} = \max_{i\in V_t} \delta_i$, and let us define the characteristic matrix P as:

$$P(G_t) := I - \frac{1}{\delta_{max}+1}(D-A) \tag{5.6}$$

The matrix P is symmetric and doubly stochastic ($\sum_i P_{ij} = \sum_j P_{ij} = 1$) by construction, which are characteristics needed by Duchi *et al.* for their convergence proof to be valid, and is an essential part of our algorithm. [123]

[3] $\|\cdot\|_*$ is the dual norm to $\|\cdot\|$. See [123] for details

Algorithm details

First we note that the distributed dual averaging is implemented at a particular fixed time t. The details described here are for iterations (counted using the parameter $k = 1,2,\ldots$) that occur at a time (t) during the evolution of the problem $\mathscr{P}_{t+}$. During these iterations, each node maintains a local copy of two values $(x_i(k), z_i(k)) \in X \times \mathbb{R}^1$, and also calculates the subgradient $g_i(k) \in \partial f_i(x_i(k))$. Each node receives information about parameters $\{z_j(k), j \in N(i)\}$ from nodes in its neighborhood. Define a projection onto X with respect to some proximal function ϕ and positive scalar step-size α as

$$\Pi_X^{\phi}(z,\alpha) := arg\min_{x\in X}\{< z,x > +\frac{1}{\alpha}\phi(x)\} \tag{5.7}$$

Given a non-increasing sequence of positive step-sizes $\alpha(k)$, the DDA algorithm involves each node $i \in V_t$ via the following iterative updates at a fixed time t :

$$\begin{aligned} z_i(k+1) &= \sum_{j\in N(i)} P_{ji} z_j(k) + g_i(k) \\ x_i(k+1) &= \Pi_X^{\phi}(z_i(k+1), \alpha(k)) \end{aligned} \tag{5.8}$$

Convergence of the DDA algorithm to the optimum $x^* \in X$, convergence rates and modifications for various network types (random, fixed degree etc.) can be found in [123]. However, this method is only suited to solve static problems, and not dynamic or evolving problems that we are interested in. Strong convergence results for even variants of DDA that include stochastic communication links and composite objective function forms, as well as superior experimental performance when compared to state-of-the-art algorithms such as Markov Incremental Gradient Descent (MIGD) [124] and Distributed Projected Gradient Method [125] encourage us to modify this algorithm suitably for this new class of evolving optimization problems. The following section introduces the Bregman Projection, which is helpful in motivating the need for our Adaptive Random Projection (ARP) algorithm in the pivot phase.

5.2.2 Bregman projection method

Of the several algorithms related to steered sequential projections described by Censor *et al.* [126], we are interested in algorithms that use Bregman distance functions for projection onto a set. A Bregman distance is defined on a zone S with respect to a function f, and denoted by $D_f(x,y)$ for the distance between two points x and y as

$$D_f(x,y) := f(x) - f(y) - < \nabla f(y), x-y > \tag{5.9}$$

If $\Omega \subseteq \mathbb{R}^n$ is closed, a Bregman projection (P^f_Ω) is then defined as

$$\Pi^f_\Omega(x) = arg\min_z \{D_f(z,x) \mid z \in \Omega \cap cl(S)\} \tag{5.10}$$

where $cl(S)$ is the closure of the zone S. We use a special form of the Bregman function $1/2\|x\|_2^2$ (with zone $\mathbb{R}^n$). Recall that this Bregman function has the same form as the proximal function used in Sect. 5.2.1. In this case, Bregman projections become orthogonal (Euclidian) projections. The iterates of the algorithm are defined as

$$x(k+1) = x(k) + \sigma_k(P_{i(k)}(x(k)) - x(k)) \tag{5.11}$$

where $\{i(k)\}$ is a cyclic control sequence, and the sequence $\{\sigma_k\}_{k\geq 0}$ is an m-steering sequence as defined in Censor *et al.* [126] However, although the steered version of the Bregman Projection method can solve inconsistent problems, it cannot be applied to evolving optimization problems where $\mathscr{P}_{t-} \neq \mathscr{P}_t \neq \mathscr{P}_{t+}$.

Given that we are able to find the optimum f_t^* of a problem $\mathscr{P}_t$ of the form in equation 5.3 through updates of the DDA algorithm (equation 5.8) at some time $t > 0$, our aim is to find a superior $f_{t+}^* < f_t^*$ by activating or deactivating certain nodes in the network ($i \in V_{t+}, V_{t+} \subseteq \{1,2,\ldots,n\}, V_{t+} \neq V_t$). In words, we are modifying the network by selecting a new set of nodes from a 'library' of n existing nodes. Solving the problem $\mathscr{P}_{t+}$ using DDA again may find an $f_{t+}^* < f_t^*$, but we are ignoring the progress made through solving the problems $\mathscr{P}_0$ to $\mathscr{P}_t$. Thus, a form of numerical continuation is required, and is established through a 'pivot' phase at time t, before solving the problem $\mathscr{P}_{t+}$.

At time t, problem $\mathscr{P}_t$ of the form equation 5.3 is solved to obtain an optimum f^* and a corresponding value of $x^* \in X$. Let $\triangle V_t$ be the symmetric difference between the sets V_t and V_{t+} (that is, $\triangle V_t = V_t \triangle V_{t+} = (V_t \cup V_{t+}) \setminus (V_t \cap V_{t+})$. This gives us the nodes added or removed in the new time step $t+$. It is important to note that typically nodes are added to the network since most applications of the algorithm (as will be seen in the penultimate section). Thus, we can re-write equation 5.3 as follows

$$\begin{aligned} &\min_x \sum_{i \in V_{t+}} f_i(x) \\ \equiv &\min_x (\sum_{i \in V_t} f_i(x) \pm \sum_{i \in \triangle V_t} f_i(x)) \\ &x \in X \end{aligned} \tag{5.12}$$

Since the first sum of functions have been solved for in the previous time step with $i \in V_t$, the problem at the pivot step is reduced to a simpler, convex feasibility problem which can be written as

$$\begin{aligned} &\min_x \mathbf{0}^T x \\ f^*_{t+1} = &\sum_{i \in V_{t+}} f_i(x) < f^*_t, x \in X \end{aligned} \tag{5.13}$$

Represent the convex set $x \in X$ at any pivot time as Q_X and $\sum_{i \in V_{t+}} f_i(x) < f^*_t$ as Q_F. Thus at the pivot point, our solution is a point $\tilde{x} \in Q_X \cap Q_F$. There exist several convex feasibility algorithms for finding a point $x^* \in Q := \cap_{i=0}^{m-1} Q_i$, which is the intersection of finitely many closed, individual sets $Q_i \in \mathbb{R}^n$. [127–129] However, most of these algorithms are applicable only to the 'consistent' case wherein $Q \neq \varnothing$. In our case, it is not known *a prori* that $Q \neq \varnothing$ at time $t+$. Thus, a sequential projection method that is proved to be convergent for the inconsistent as well as consistent case is used here. [126] Thus, we are interested in an iterative algorithm to find a point in the intersection of the sets Q_X and Q_F. Additionally, as in the paper by Nedic, we would also like to determine if such a point exists by observing the algorithm's iterates itself. [120] Our Adaptive Random Projections (ARP) algorithm (introduced and analyzed in the following sections) is well suited to solve such dynamic problems.

5.3 Adaptive Random Projection

Our algorithm is based on the hypothesis that finding a point in the intersection of Q_i's corresponds to a an improvement in f^*. We prove this hypothesis using the following argument.

Consider an existing $f^* \geq \sum_i f_i(x) = F(x)$, and a pair of new candidate functions f_1 and f_2. First suppose that adding f_1 does not satisfy the feasibility criterion in Equation 5.13, and that adding f_2 does satisfy it. In other words, a) $F(x) + f_1(x) - f^* > 0 \; \forall x \in Q_1$ but b) $F(x) + f_2(x) - f^* \leq 0 \; \forall x \in Q_2$. We are interested in the case where both these conditions a) and b) are simultaneously true, that is $Q_1 \cap Q_2 \neq \varnothing$. Exclusion of the node corresponding to f_1 is justified since $f^* \geq F \;\; \forall x \in Q_1$, and $F(x) + f_1(x) + f_2(x) - f^* \leq f_1(x) \;\; \forall x \in Q_1 \cap Q_2$. Of course, when $Q_1 \cap Q_2 = \varnothing$, we cannot find a common point x to evaluate f_1 and f_2.

Now consider another candidate function f_3 which satisfies c) $F(x) + f_3(x) - f^* \leq 0$ $\forall x \in Q_3$. We are interested in the scenario where b) and c) are simultaneously true. When $Q_2 \cap Q_3 \neq \varnothing$, $F(x) + f_2(x) + f_3(x) - f^* \leq f_2(x)$ and $F(x) + f_2(x) + f_3(x) - f^* \leq f_3(x)$ are both true. Since $f_2 \leq f^* - F(x)$ and $f_3 \leq f^* - F(x)$, $\{F(x) + f_2(x) + f_3(x) - f^*\} \leq 0$ always holds true. Also, there may be a greater benefit when both functions f_2 and f_3 are considered simultaneously than with any one of the functions alone. More nodes with corresponding functions f_i can be added when b) and c) are found to be true. With this motivation, we now proceed to introduce our algorithm.

5.3.1 Algorithm details

Recall that in Section 5.2.2, we introduced our problem as a convex feasibility problem. More precisely, our problem is related to a body of research on the Maximum Feasible Subset (MFS) problem in which the goal is to identify (enumerate) the sets that have a feasible intersection. In view of our overall goal, a feasible intersection of a subset of the candidate library of nodes translates to a definite improvement of the global objective function by adding these nodes. Other similar but closely related problems include the Minimum Unsatisfied Linear Relation problem (MINULR) and Minimum Cardinality Set

Covering Problem (MINCOVER) that are typically applied to a set of linear constraints. Finding the MFS of linear constraints is NP-hard, and is widely studied. [130–132]

In the pivot stage, we are to determine a point x^* such that it lies in the intersection of a select subset of $x^* \in X \triangleq \bigcap_{i \in \mathscr{I}} X_i$, where $\mathscr{I} \in 1, \ldots, m,\ |\mathscr{I}| \geq 2$. Our proposed Adaptive Random Projection algorithm is stated as follows - given an iterate $x_k \in \mathbb{R}^n$, subsequent iterates are given by

$$x_{k+1} = \Pi_{\omega_k}(x_k) \tag{5.14}$$

where the random variable ω at time step k is a value in the set $\{1, \ldots, m\}$. Each constraint set X_i is associated with an "agent" or "node" that decides which set to project to next by maintaining a set of m probabilities. Each of these probabilities corresponds to projecting onto a set i. Denote this set of m probabilities associated with the node i at an iteration k as Pr so that $Pr(i, j)$ implies probability of projecting from set i to set j. Note that traditionally in literature, only probability of projecting onto a set $Pr(i)$ is defined. Here we can obtain this value implicitly, that is $Pr(i) = \sum_{j=1}^{m} Pr(j, i)$.

We identify the Maximum Feasible Set (MFS) during the course of convergence by using constraint violation values. Constraint violation with respect to a constraint set i can be calculated (using Equation 5.13) as

$$c_i = \max \left(\sum_{i \in V_{t+}} f_i(x) - f_t^*, 0 \right) \tag{5.15}$$

Define $d_X(x)$ Bregman distance of a point $x \in \mathbb{R}^n$ to a set X. Notice that a point on the intersection of the MFS does not correspond to a point with the minimum distance to all sets d_{min}. That is, we are interested in the case where

1. $d \neq 0$, since all the sets in question (X_i's) may not intersect
2. $d \neq d_{min}$, since some sets may be excluded from the MFS, and
3. $d = d^* > d_{min}$ which is corresponds to a point on the intersection of the MFS (see Figure 5.1)

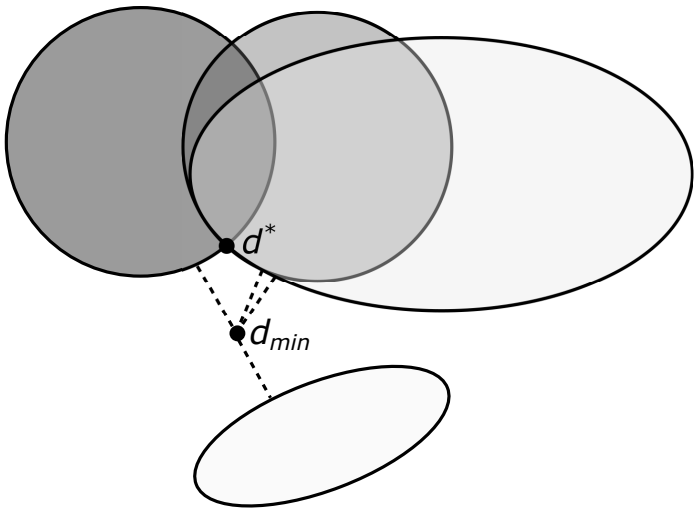


Figure 5.1.: We intend to find d^* at the intersection of the MFS rather than d_{min} which applies to all four sets shown in the figure

Let c_{ij} be the constraint violation measured using equation 5.15 for set i from a point on set j. We are now ready to define the adaptive probabilities $Pr(i,j)$ for iterations $k = 0,1,2,\ldots$

$$Pr(i,j) = \begin{cases} 0 & \text{if } i = j,\ k = 0 \\ 1 - \sum_{j \neq i} Pr(i,j) & \text{if } i = j,\ k > 0 \\ \frac{2}{m-1}\left(1 - \frac{1}{1+\exp(-\gamma_k \bar{c}_{ij})}\right) & \text{if } i \neq j,\ k \geq 0 \end{cases} \tag{5.16}$$

where $\bar{c}_{ij}$ is the moving average value of c_{ij}, and γ_k is a gradually increasing sequence of positive numbers that amplify the constraint violation $\bar{c}_{ij}$ with the properties $\gamma_{k+1} > \gamma_k$, $\gamma_0 = 0$ and $\sum_{k=0}^{\infty} \gamma_k = \infty$. Pr is a doubly stochastic matrix like P, that is used in the first phase (Equation 5.6). Note that at iteration $k = 0$, $Pr(i,j) = \frac{1}{m-1} \forall i \neq j$. Thus, it is equally probable to project onto other sets j from set i. As the iterations progress, the algorithm is expected to initially behave like a standard random projection algorithm to find d_{min} since γ_k is close to zero. As the average constraint violation $\bar{c}_{ij}$ increases, it becomes less probable to project onto the set j from set i. Also, a stagnant or non-changing constraint violation

is also penalized since γ_k keeps increasing. This reduces the probability of projecting onto frequently penalized nodes gradually towards zero.

An easy way to visualize the above paragraph is as follows. Let the candidate nodes to be added $i = 1, \ldots, m$ be a fully connected graph $G = (V, E)$. Initially, all links exist, and we wish to gradually fade away these links as $\bar{c}_{ij}$ increases. As the algorithm progresses, $Pr(i, j) \to 0$ implies that link does not exist. A high probability $Pr(i, j)$ and $Pr(j, i)$ indicates that sets i and j may have an intersection. On the other hand, a high value of $Pr(i, j)$ and a corresponding low probability of $Pr(j, i)$ implies that improvement of f^* is relatively higher for set corresponding to j than for set i. In this sense, ARP is an online learning algorithm. The network analogy will be referred to again since it is useful for our next step, and also lends support to practical implementation of the algorithm on actual distributed nodes.

5.3.2 Continuation via Edge Contracting

As discussed in the previous paragraph, edge contracting plays an important role in ARP. Recall that a link between two nodes implies that an intersection probably exists. Since the above procedure aims to isolate unconnected nodes while confirming pairwise intersections, we continue our analysis by using an edge contracting procedure. This is visually similar to the Karger's algorithm (See figure 5.2), but has a completely different purpose and procedure. [133] In our scheme, the result of 'contracting' an edge that connects nodes u and v is the formation of a new node $u - v$. Since $Pr(i, j)$ is not necessarily symmetric, we obtain a directed version of the graph shown in Figure 5.2. At this point, we are convinced that an intersection exists between the sets X_u and X_v. Denote the set $X_{u-v} = X_u \cap X_v$, as the intersection set and the corresponding node on our graph $G = \{V, E\}$ as $u - v$. The procedure is formally described below:

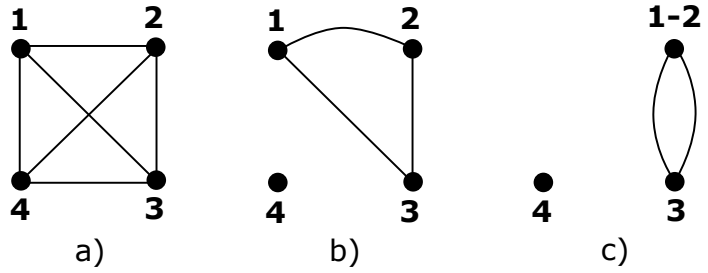


Figure 5.2.: a) Initialized graph with respect to the hypothetical problem in figure 5.1. b) Node 4 corresponding to set 4 is isolated as part of ARP. c) Node 1 and node 2 coalesce to form node 1-2. Projections are thereafter performed one node 1-2 and 3

Procedure Contract $G = \{V, E\}$

IF $|V| \geq 1$

Choose edge $e = \{u, v\} \in E : Pr(u, v) > 0 \ \wedge \ Pr(v, u) > 0 \ \wedge \ \gamma_k \bar{c}_{uv} \to \infty$

Denote new node $u - v$

$G = G \setminus e$

$V = V \setminus \{u, v\}, V = V \cup u - v$

Isolate nodes:

Choose node $k \in V : Pr(u, k) = 0 \ \wedge \ Pr(k, u) \geq 0 \ \ \forall u \neq k, \ u \in V$

$V = V \setminus k$

$\gamma_{k+1} = \gamma_0$

Continue ARP

Practically, we look for a $Pr(u, v) > P_{thr}$ instead of $Pr(u, v) > 0$, and $Pr(u, v) < Pr_{thr}$ instead of $Pr(u, v) = 0$, where P_{thr} is some small positive threshold value. Note that a node may be isolated if all edges to it, or from it are removed. After the formation of the node $u - v$, all projections are made through parallel projections onto both sets Π_{X_u} and Π_{X_v}. As a result, projections onto the set X_{u-v} may be obtained using:

$$\Pi_{X_{u-v}}(x) = \frac{\Pi_{X_u}(x) + \Pi_{X_v}(x)}{2} \tag{5.17}$$

Bauschke and Borwein study the convergence of these parallel (or more generally, weighted) projections through the use of active indices in their review paper. [134] Thus, in our algorithm we simply consider the average projection as the projection onto node $u-v$. Note that as the algorithm progresses along with edge contracting, we coalesce nodes with feasible intersections into a node denoted by $u-v-\cdots-w$. Finally, the desired MFS is obtained by the indices that identify this coalesced node. Note that in some cases, the coalesced node(s) obtained may be strongly associated with other nodes and as such it may become necessary to include these nodes into the final solution.

5.3.3 Mathematical Convergence of ARP

In this section we present mathematical convergence results in relation to our algorithm. Our goal is to show that in the pivot stage,

$$\|x_{k+1} - x^*\| \leq \|x_k - x^*\| + e(x_k, \omega_k) \tag{5.18}$$

As in Wang and Bertsekas [135], we show this by bounding each term on the right hand side (RHS) of the above equation to show that the iteration error e, which is a function of the current iterate x_k and the random index ω_k is stochastically decreasing.

To do this, we take the conditional expectation of the "average improvement" of the process given its history $\mathscr{F}_k = \{x_0, x_1, \ldots x_k, \omega_0, \omega_1, \ldots \omega_k, \gamma_0, \gamma_1, \ldots, \gamma_k\}$. That is, we analyze $E[\|x_{k+1} - x^*\|^2 \,|\mathscr{F}_k]$ where $E[\cdot]$ denotes the expected value of a random variable.

Finally we study the equivalence of two problems - one involving our version of MFS involving all m sets, and the second involving convergence onto a smaller number of sets which are results of solving the MFS problem *a priori*. As in Wang and Bertsekas, we assume that the collection of sets X_i satisfy linear regularity as follows.

Assumption 1 *There exists a positive scalar η such that for any $x \in \mathbb{R}^n$*

$$\|x - \Pi(x)\| \leq \eta \max_{i \in V : G = \{V,E\}} \|x - \Pi_{X_i}(x)\|$$

There are several situations including polyhedral sets where linear regularity holds true, and for a detailed discussion on this property we refer the reader to Deutsch and Hundal. [136] In simple words, the assumption roughly translates to saying that the distance of a point from two sets is closely related to the distance from the intersection of these two sets, when such an intersection exists.

We also assume non-expansiveness of the projection operator Π, that is:

Assumption 2 *For any two points x and y in $\mathbb{R}^n$, and for all projections $\Pi = \Pi_{X_i}$ considered*

$$\|\Pi(x) - \Pi(y)\| \leq \|x - y\|$$

Finally assume that we can generate an increasing sequence of numbers $\{\gamma_k\}$. With this, we begin our mathematical treatment of the algorithm, following the stencil provided by Wang and Bertsekas. [135].

Lemma 1 *For any $x \in \mathbb{R}^n$, and $y \in S$ with $S \subseteq \mathbb{R}^n$,*

$$\|\Pi_S(x) - y\|^2 \leq \|x - y\|^2 - \|x - \Pi_S(x)\|^2$$

Proof .

$$\begin{aligned}
\|\Pi_S(x) - y\|^2 &= \|\Pi_S(x) - x + x - y\|^2 \\
&= \|x - y\|^2 + \|x - \Pi_S(x)\|^2 - 2 \cdot (x - y)'(x - \Pi_S(x)) \\
&= \|x - y\|^2 + \|x - \Pi_S(x)\|^2 - 2 \cdot (x - \Pi_S(x) + \Pi_S(x) - y)'(x - \Pi_S(x))
\end{aligned}$$

Since $(y - \Pi_S(x))'(x - \Pi_S(x)) \leq 0$

$$\begin{aligned}\|\Pi_S(x)-y\|^2 &\leq \|x-y\|^2+\|x-\Pi_S(x)\|^2-2\cdot(x-\Pi_S(x))'(x-\Pi_S(x))\\ &= \|x-y\|^2-\|x-\Pi_S(x)\|^2\end{aligned}$$

■

For the next Lemma, we use the following information:

1. Lemma 1
2. ARP iterates $x_{k+1}=\Pi_{\omega_k}x_k$
3. $2a'b\leq\varepsilon\|a\|^2+\frac{1}{\varepsilon}\|b\|^2\,\forall a,b\in\mathbb{R}^n$ [135]
4. Distance $d_X(x)=\|x-\Pi_X(x)\|$

Lemma 2 *For any $x\in\mathbb{R}^n$, and $y\in S$ with $S\subseteq\mathbb{R}^n$,*

$$\|x_{k+1}-y\|\leq(1+\varepsilon)\|x_k-y\|+(1+\frac{1}{\varepsilon})d^2(x_k)$$

Proof .

$$\begin{aligned}\|\Pi_S(x)-y\|^2 &\leq \|x-y\|^2+\|x-\Pi_S(x)\|^2-2\cdot(x-y)'(x-\Pi_S(x))\\ &\leq \|x-y\|^2+\|x-\Pi_S(x)\|^2+[\varepsilon\|x-y\|^2+\frac{1}{\varepsilon}\|x-\Pi_S(x)\|^2]\\ &= (1+\varepsilon)\|x-y\|^2+(1+\frac{1}{\varepsilon})\cdot d^2(x)\end{aligned}$$

■

where the required result is easily obtained by substituting $x=x_k$. A consequence of this lemma can be seen by substituting $x=x_k$ and $y=x^*$, and having the set $S=X_{\omega_i}$, that is

$$\begin{aligned}&\|\Pi_{\omega_i}(x_k)-x^*\|^2\\ &=\|x_{k+1}-x^*\|^2\leq(1+\varepsilon)\|x_k-x^*\|^2+(1+\frac{1}{\varepsilon})\cdot d^2(x_k)\end{aligned}\tag{5.19}$$

Although this is not of significant consequence to our "expected" or average decrease in error, it is an interesting side note. Also, we can extend this result to an N step look-ahead as follows. We are interested to study the progress of the quantity $\|x_{k+N} - x^*\|^2$ as N grows. Equation 5.19 looks one step ahead, that is, $N = 1$. For $N = 2$,

$$\begin{aligned} \|x_{k+2} - x^*\|^2 &\leq \|x_{k+1} - x^*\|^2 - d^2(x_{k+1}) \\ &\leq \|x_k - x^*\|^2 - d^2(x_k) - d^2(x_{k+1}) \end{aligned} \tag{5.20}$$

Therefore, for an N step look-ahead we can construct Equation 5.21:

$$\|x_{k+N} - x^*\|^2 \leq \|x_k - x^*\|^2 - [d^2(x_k) + d^2(x_{k+1}) + \ldots + d^2(x_{k+N-1})] \tag{5.21}$$

5.3.4 Progress Towards the Optimum

Before progressing to proposition 1, we provide a lower bound of the ARP. This average progress of ARP is given as the expected value of $\|x - \Pi_{\omega_k}(x)\|$ subject to the history of the algorithm's iterates $\mathscr{F}_k$ until the current iteration k.

$$\begin{aligned} E[\|x - \Pi_{\omega_k}(x)\|^2 | \mathscr{F}_k] &= \sum_{i \in V} Pr(\omega_k = i | \mathscr{F}_k) \|x - \Pi_i(x)\|^2 \\ &= \sum_{i \in V} Pr(\omega_k = i | \omega_{k-1} = j) \|x - \Pi_i(x)\|^2 \\ &\geq \frac{2}{m-1} \left(1 - \frac{1}{1 + \exp -\gamma_k \bar{(c)}_{ij}}\right) \|x - \Pi_i(x)\|^2 \end{aligned} \tag{5.22}$$

where $\bar{c}_i j$ corresponds to the minimum corresponding average constraint violation. Now maximizing the RHS over j and using linear regularity (see assumption 1)

$$E[\|x - \Pi_{\omega_k}(x)\|^2 | \mathscr{F}_k] \geq \frac{2}{\eta(m-1)} \left(1 - \frac{1}{1 + \exp(-\gamma_k \bar{c}_{ij})}\right) \|x - \Pi_i(x)\|^2 \tag{5.23}$$

Finally, we discuss the average progress of the algorithm towards the solution x^* in the following proposition.

Proposition 1 *Let assumptions 1 and 2 hold, and let x^* be an optimal solution such that it converges to the MFS intersection while identifying the set itself. Then the ARP algorithm generates a sequence of iterates xk such that*

$$E[d^2(x_{k+1})|\mathscr{F}_k] \leq \left(2+\varepsilon+\frac{1}{\varepsilon}\right)\cdot\frac{1}{\eta(m-1)}\cdot d^2(x_k)$$

Proof . Let ε be a positive scalar as defined earlier. Let d represent the distance to the intersection of the MFS, and S represent the MFS. In Lemma 2, we use $y=\Pi_{\omega_k}x_k$ and $d^2(x_{k+1}) \leq \|x_{k+1}-\Pi_S(x_k)\|^2$ to get

$$\begin{aligned} d^2(x_{k+1}) &\leq \|x_{k+1}-\Pi_S(x_k)\|^2 \\ &\leq (1+\varepsilon)\|x_k-\Pi_S(x_k)\|+(1+\frac{1}{\varepsilon})d^2(x_k) \end{aligned}$$

Now, taking the conditional expected value of both sides with respect to $\mathscr{F}_k$ and using Equation 5.23, (or see Equation 5.19) we get

$$\begin{aligned} E[d^2(x_{k+1})|\mathscr{F}_k] &\leq (2+\varepsilon+\frac{1}{\varepsilon})\cdot\frac{2}{\eta(m-1)}\cdot\left(1-\frac{1}{1+\exp(-\gamma_k\bar{c}_{ij})}\right)\cdot d^2(x_k) \\ &\leq (2+\varepsilon+\frac{1}{\varepsilon})\cdot\frac{1}{\eta(m-1)}\cdot d^2(x_k) \end{aligned}$$

■

5.3.5 Implication of γ_k sequence

The purpose of γ_k is to gradually amplify the effect of the average constraint violation $\bar{c}_{ij}$. For the purpose of this discussion, let us assume that the sequence of $\{\gamma_k\}$ is generated using:

$$\gamma_{k+1} = r_\gamma \cdot \gamma_k \tag{5.24}$$

with $\gamma_0 > 0$, and rate of increase $r_\gamma > 0$. It is obvious that the rate r_γ must be upper-bounded to avoid artificial or premature convergence. This may lead to a wrong choice of nodes that

form the MFS, although some node that improves the value of f^* may be found. However, the value of this upper bound depends on the rate of variation of the average constraint violation $\bar{c}_{ij}$ which is problem specific.

5.4 Application problems

We now explore two application problems 1) a topology optimization problem solved using a centralized optimization routine to find an initial f^* followed by ARP (Section 5.4.1, and 2) a sensor management problem solved using DDA to find the initial f^* followed by ARP (Section 5.4.2). A modified version of the application problem on topology optimization will be used in Chapter 6 in the aero-structural optimization module. The second application problem in this chapter is provided to highlight the the fact that a variety of problems (even ones that include uncertain parameters) may be solved using ARP.

5.4.1 Application to a Topology Optimization problem

The purpose of topology optimization is to find the optimum layout of a structure given the domain, loads and distribution of material. The Messerschmitt-Bolkow-Blohm (MBB) beam problem is a classical problem in topology optimization literature and is used as an example case here. [137] The objective is to minimize compliance with the constraint on the amount of material used. The standard version of the MBB problem is interesting in itself, but has a fixed design domain boundary. Our implementation can be easily extended to increasing domain sizes. In our version of the problem, four agents or nodes control the extended portions of the the design space as shown in the figure, each of which contain 15 additional variables (See Figure 5.3).

Figure 5.3.: The design domain, boundary conditions, external load applied, and new portions of the extended domain that the four new agents are in control of (marked 1,2,3 and 4)

Formally stated, the problem we wish to solve is:

$$\begin{aligned} \min_x c(x) &= U^T K U = \sum_{i=1}^{N} E_i(x_i) \cdot (u_i)^T k u_i \\ \text{S.To. } & V(x)/V_0 = v_{frac} \\ & KU = F \\ & 0 \leq x \leq 1 \end{aligned} \tag{5.25}$$

where $c(x)$ is the compliance of the structure, which is a global measure of deformation, U, F and K are the global displacement vector, force vector and stiffness matrix for an element with unit Young's modulus respectively. $V(x)$ and V_0 are the material volume and domain volume, and v_{frac} is the volume fraction. For a given domain size, v_{frac} is fixed. This value is adjusted to maintain the total amount of material used as we evolve the domain. The objective function is written as a sum of compliance of individual elements with u_i being the displacement vector of each element, and k being the element specific stiffness matrix. It is assumed that the Young's modulus of each element is obtained as a function of densities x_i according to Equation 6.27 shown below.

$$E_i(x_i) = E_{min} + x_i^p (E_0 - E_{min}) \tag{5.26}$$

The values of E_{min}, E_0 and p used here are $1e^{-9}$, 0 and 3 respectively. More information about the optimality criteria method, sensitivity studies and alternative solution methodologies may be obtained from a wide variety of existing literature. [137–142] It is important to note that the material considered here is isotropic and linearly elastic. Compared to the optimality criteria method, other methods such as the Level-set method, or Finite Element Method (FEM) may provide higher fidelity results and/or reduced noise and checkerboxing.

A significantly modified version of the 88 line topology optimization code by Sigmund is used to find the optimum f^* and to perform projections. [139] Note that the strategy of implementing the projections as an optimization problem that minimizes the norm is not practical here since the total number of variables involved is very large (1260 to be exact). Thus the projections used here may be non-orthogonal, but suit the purpose of the paper since the projections are extremely efficient.

Results

All simulations were conducted in a custom MATLAB 2014a code running on an Intel Core i7-2630QM 2GHz processor. The sequence of γ_k was generated as a simple geometric sequence with $\gamma_0 = 0.01$ and $\gamma_{k+1} = \gamma k * 1.0015$ (refer Section 5.3.5 for discussion on rate of increase of γ). Optimization using a heavily modified version of [139] yielded the layout and structure shown in Figure 5.5a. The maximum number of iterations in the pivot phase is set to be 5000, and is sufficiently large as discussed later in this section. Since we solve a feasibility problem using alternating projections in the pivot phase, agents do not typically have access to the overall objective function $c(x)$. Hence, during the pivot phase, we use a density filter that directly transforms the original variables (x) rather than a sensitivity filter as used in the first phase which modifies the sensitivities $\frac{\partial c}{\partial x}$ (see details of implementation in Sigmund [138, 139]).[4]

[4]Note that solving the first phase of this problem (i.e. finding f^*) using the original DDA is impractical due to the number of agents/nodes required, and is also irrelevant for this problem type.

We stop the iterations in the pivot stage if the maximum number of projections of 5000 is reached (implying usability in on-board systems) or if a probability value in $Pr(i,j)$ matrix is above 0.95. Also, for coalescing nodes, we use the a threshold probability value of $Pr_{thr} = 0.01$. The average constraint violation matrix $\bar{c}_{ij}$ at the end of 4723 projections is shown below:

$$\bar{c}_{i,j} = \begin{pmatrix} 0 & 1.1324 & 0.7900 & 0.5672 \\ 0.5769 & 0 & 1.1447 & 0.5107 \\ 0.5679 & 0.8067 & 0 & 0.9616 \\ 0.8339 & 0.7300 & 0.6129 & 0 \end{pmatrix}$$

$$\Pr(i,j) = \begin{pmatrix} 0.9725 & 0.0007 & 0.0056 & 0.0210 \\ 0.0199 & 0.9502 & 0.0007 & 0.0292 \\ 0.0209 & 0.0051 & 0.9720 & 0.0020 \\ 0.0043 & 0.0080 & 0.0161 & 0.9716 \end{pmatrix}$$

Given the value of P_{thr} used, we obtain the following directed graph corresponding to $Pr(i,j)$ values shown in .

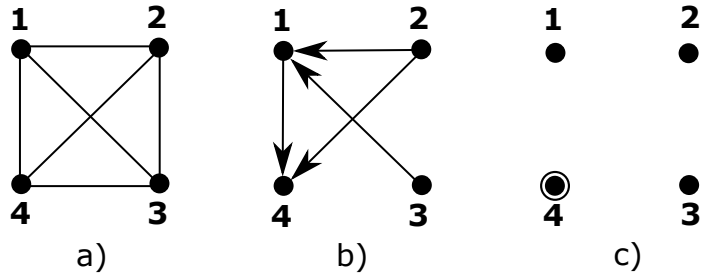
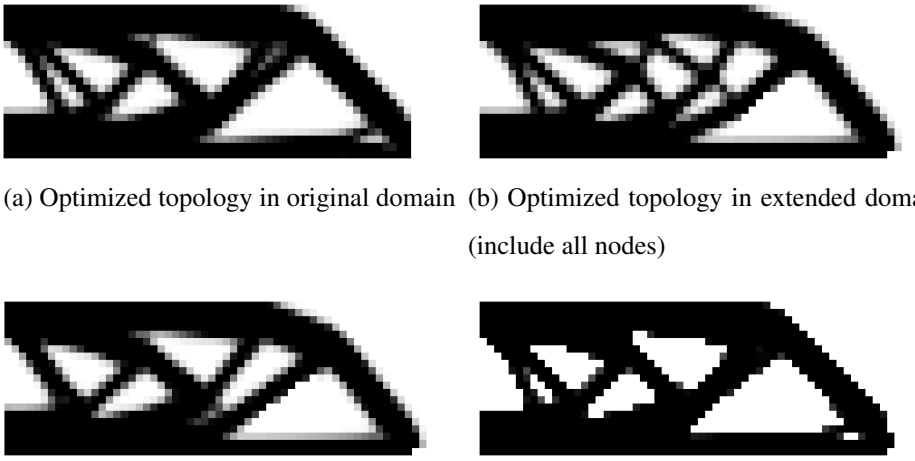


Figure 5.4.: Edge contracting procedure for topology optimization example. As a result, node 4 is selected, and all other nodes are isolated.

Thus the MFS of this problem is set 4 alone, and the corresponding sub-domain is added to the topology optimization problem. We then continue optimizing the topology to obtain the final structure. Four differentiating cases are tested as shown in figure **??**.

The optimum topology obtained in the original domain is shown in Figure 5.5a. Solving the problem using the entire extended domain (all four nodes included) in a centralized manner yields the result shown in Figure 5.5b. Figure 5.5c is the topology obtained when node 4 (which is ascertained as a result of ARP) is added to the original domain *a-priori*, and solved in a centralized manner. Figure 5.5d represents the solution method presented in this paper, that is if the topology obtained in Figure 5.5a is used after the ARP pivot phase until convergence.



(a) Optimized topology in original domain

(b) Optimized topology in extended domain (include all nodes)

(c) Optimized topology in partially extended domain (include node 4)

(d) Final topology if domain is improved using ARP

Figure 5.5.: Comparison of four cases tested - The actual procedure corresponds to improving structure in a) using ARP to obtain d)

As seen in Table 5.1, Case 5.5d reports the best final f^* and only requires an additional 9 iterations to converge after the pivot phase. The centralized cases 5.5b and 5.5c also converge to better values when compared to the original. The topologies obtained in the four cases reported here are vastly different (visually) when compared to the relative closeness in f^* values. It is interesting to note that when node 4 is forcibly added to the domain (Case 5.5c), the structure obtained is different as the path to that optimum is different. Similar looking topologies were obtained in literature using other methods. The results

presented herein are repeatable and may be obtained using codes and input parameters provided in [138, 139] corresponding to the MBB-beam problem.

Case	Figure	Iterations	f^*
Case a	5.5a	129	120.0345
Case b	5.5b	158	119.9221
Case c	5.5c	237	119.3652
Case d	5.5d	138	115.5444

Table 5.1: Comparison of the four test cases in terms of number of iterations and f^*.

5.4.2 Application to a Sensor Management problem

There has been a growing interest in topics related to distributed optimization studies of sensor management applications. Generally, optimization involves improving the value of some system performance metric such as information maximization or risk minimization. [143] We include an application problem in the sensor management domain due to typical problem features that make application of our algorithm to this problem very apt such as 1) distributed/localized function evaluations, 2) optimization of a global goal by using local interactions, 3) the need to add additional sensors/nodes/targets to the network, each of which are typically associated with a local function, and 4) incremental algorithms like ours may be used for practical problems that involve on-board computing.

Robust estimation is a popularly solved sub-problem in this category. Each sensor in a network may collect local readings of an environmental variable (like temperature or rainfall), or of a location parameter (like an energy source or target) subject to some noise. [122, 144–151] Robust estimates of parameters and locations are often obtained from functions such as squared error and the Huber loss function [146]. Here we will make use

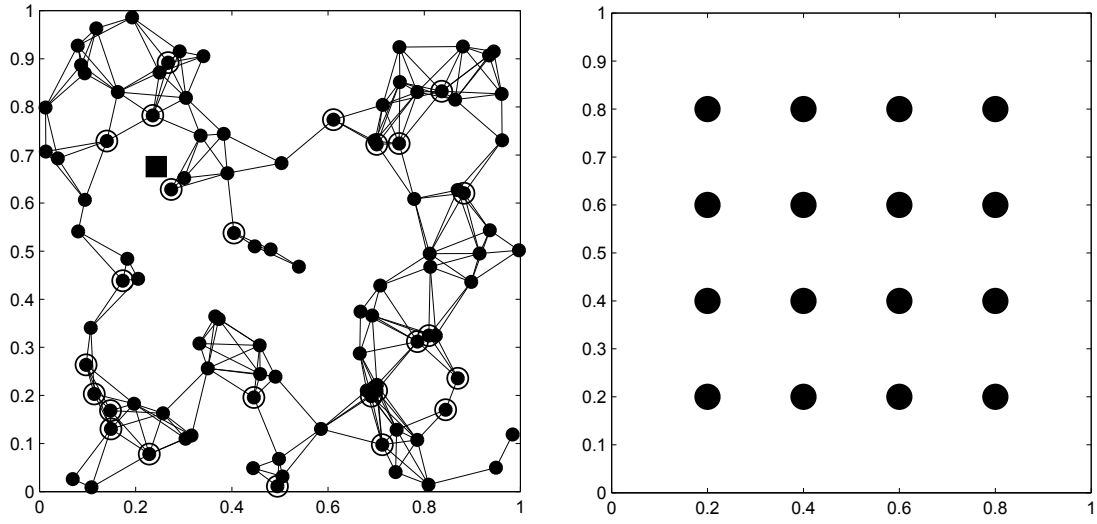
of the latter function as the local f_i being calculated at by each sensor. The Huber loss function is given by:

$$\rho(\theta,x) = \begin{cases} \frac{(x-\theta)^2}{2} & \text{if } |x-\theta| \leq \gamma \\ \gamma|x-\theta| - \frac{\gamma^2}{2} & \text{otherwise} \end{cases}. \tag{5.27}$$

with the multi-dimensional variant involving the sum of these losses across all dimensions. The Huber loss function is convex and differentiable. A general description of the robust source estimation follows. Each sensor $i = 1,\ldots,N$ collects a set of m measurements x_i, which are randomly sampled from normal distributions $\mathcal{N}(\theta,\sigma^2)$. 75% of the sensors have sample the source location θ with a noise characterized by σ^2, whereas the other 25% are faulty sensors with a noise of $10\sigma^2$. We would like to find an estimate of the source $\theta \in X$ which minimizes :

$$\begin{aligned} f(\theta,x) =& \frac{1}{n}\sum_{i=1}^{n}\rho_i(\theta,x) \\ & x \in X \end{aligned} \tag{5.28}$$

In our problem, we randomly generate a network with $N = 100$ nodes uniformly distributed on the unit square $[0,1]\times[0,1]$ (X) as shown in the figure 5.6a. We then connect the nodes based on whether their distance is less than 0.145 units (obtained by reducing the threshold value until the network remains connected) similar to the example in [152]. In the context of our paper, any new sensors added will also follow the same rule of establishing connections with other existing nodes. This relates to the real world situation where wireless sensors may connect to other sensors within some range. Each sensor processes $m = 200$ readings. The location of the sensors and the source are randomly generated. A set of 16 new candidate sensors (positions shown in figure 5.6b) are given access to the last 50 readings of the original set of 100 sensors. We first solve the problem with the original set of 100 sensors in the randomly generated network using DDA. The aim is then to use the best source location obtained as a result of the DDA algorithm to improve f^* value while simultaneously selecting the most valuable, maximum subset of the 16 new candidate sensors.



(a) Randomly generated initial network

(b) New candidate sensors

Figure 5.6.: In both images shown above, ● represents a sensor, ■ represents the source's location, and a ○ around a sensor indicates that it is faulty (also randomly generated).

Results

All simulations were conducted using a similar set-up as in section 6.7.3, but is repeated here for the sake of completeness. A custom MATLAB 2014a code running on an Intel Core i7-2630QM 2GHz processor was used. The sequence of γ_k was generated as a simple geometric sequence with $\gamma_0 = 1$ and $\gamma_{k+1} = \gamma k * 1.0015$ (refer Section 5.3.5 for discussion on rate of increase of γ). The value of α for the initial run of the DDA algorithm is fixed at a value of 0.01. The path taken by the mean source location is shown in figure 5.7. As we can see, the mean estimate of the source location continuously improves towards the actual location as the DDA algorithm progresses.

The maximum number of projections in ARP is set at 5000 (implying usability in on-board systems) or the number of projections for a probability value in $Pr(i, j)$ matrix to be above 0.95. Also, for coalescing nodes, we use the a threshold probability value of $Pr_{thr} = 0.05$. Corresponding to the $Pr(i, j)$ values, we obtain characteristic directed graphs for the

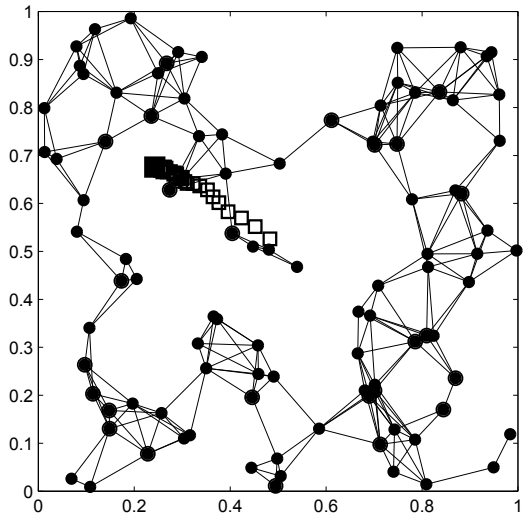


Figure 5.7.: Path taken by the mean estimate of the source (shown as □) moving towards the actual source location ■ as time progresses in the presence of faults. Sizes of the sensors ● have been reduced for clarity.

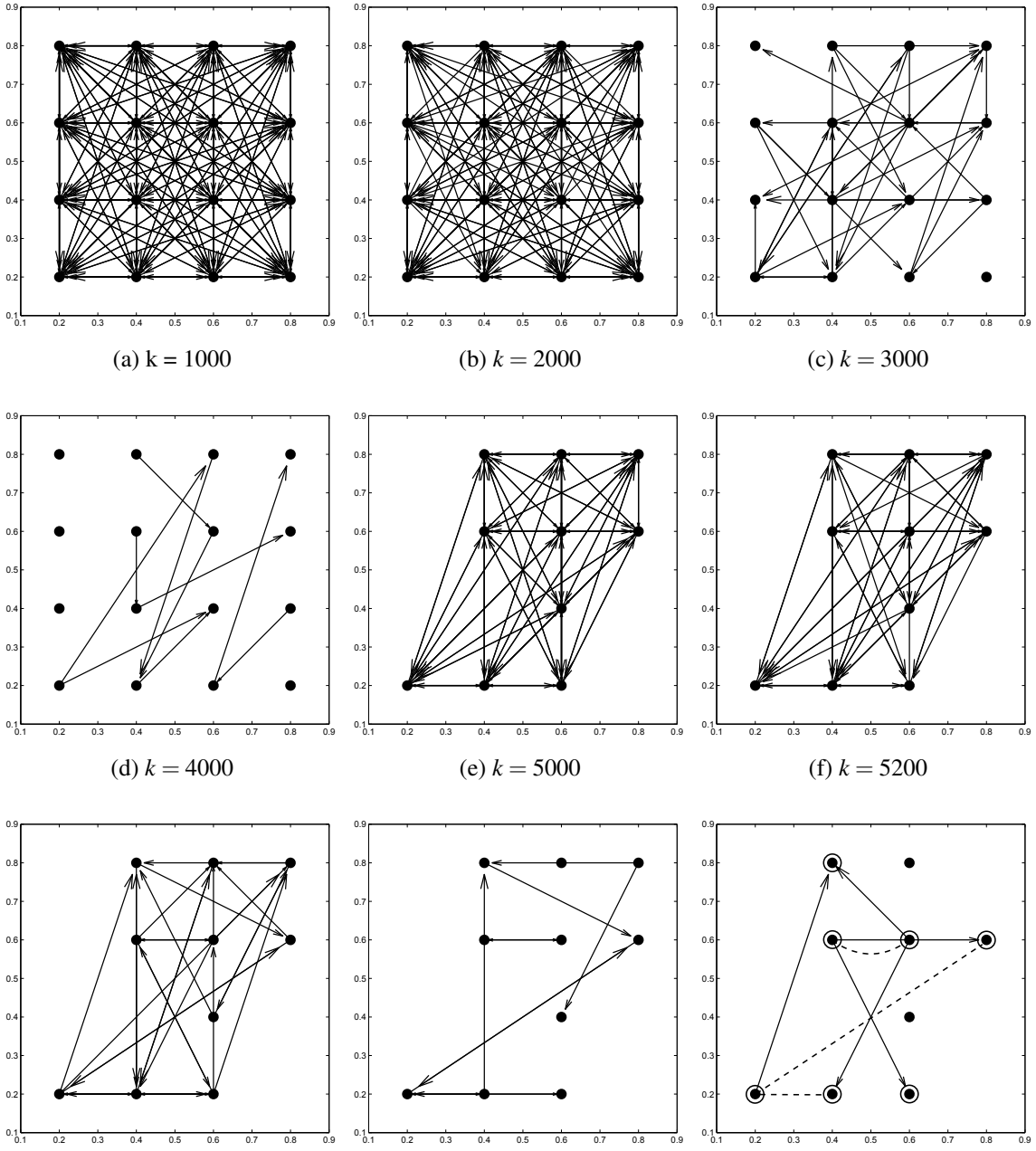
purpose of edge contraction and isolation as shown in figure 5.8. After the 5000^{th} iteration edge contraction and isolation procedures take place before ARP continues (Note transition between figures 5.8d and 5.8e). At $k = 5600$ (figure 5.8h), four pairs of nodes (seen with bidirectional links) coalesce to form new nodes while allowing the corresponding edges to contract. In figure 5.8i we see the result of ARP with a final converged set of nodes. A clear improvement in f^* is noticed while selecting candidate sensors (see table 5.2). The final network is shown in figure 5.9b.

As in section 6.7.3, we compare these results (mean f value) with a centralized optimization counterpart with the following cases - Case a) Original 100 node problem, Case b) All 16 candidate nodes are added to the network (figure 5.9a, Case c) Nodes selected by ARP are added a-priori (figure 5.9b), and Case d) Solved using DDA and ARP. Since the mean f value itself oscillates at each iteration in this example, the best and worst f values are also reported.

Case	Mean f	Std. dev f	Best f	Worst f
Case a	0.1376	0.2759	1.356e-4	1.020
Case b	0.1146	0.2840	9.839e-5	1.654
Case c	0.1135	0.2329	1.556e-4	1.264
Case d	0.1363	0.2719	2.191e-4	1.017

Table 5.2: Comparison of the four test cases in terms of number of mean, standard deviation, best and worst f values (Iterations are not reported since they are related to the number of readings taken, which in this case is always 250)

It is important to remember that Cases *a*, *b* and *c* are run with varying numbers of sensors, and therefore sample and distribute information differently. Furthermore, these cases are run for exactly 200 time steps (or iterations of the DDA algorithm). Case *d*, on the other hand includes the pivot phase of the ARP algorithm for sensor selection and improvement. The values reported in columns of table 5.2 are valid at the 200^{th} iteration. It is possible for other iterations to potray different comparisons since the problem is stochastic in nature. That being said, the ARP algorithm selects useful sensors and improves f^*. These additional 7 sensors, when added to the network *a priori* (Case *c*, figure 5.9b) and solved using DDA reports the best mean f value. When all 16 sensors are added (Case *b*, figure 5.9a), the mean, best and worst values are slightly inferior to Case *c* although this may be an artifact of the aforementioned stochastic nature of the problem.

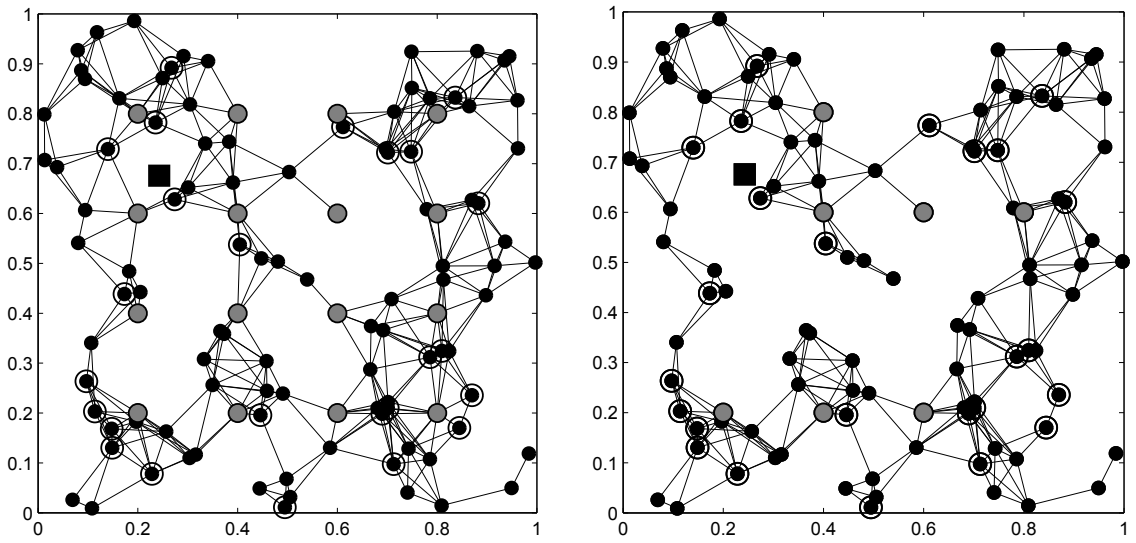


(a) k = 1000 (b) $k = 2000$ (c) $k = 3000$

(d) $k = 4000$ (e) $k = 5000$ (f) $k = 5200$

(g) $k = 5400$ (h) $k = 5600$ (i) $k = 5821$

Figure 5.8.: Progress of the ARP algorithm for selection of a subset of the 16 candidate sensors shown edge contraction of a directed graph corresponding to $Pr(i, j)$ values after the k^{th} iteration number. Circled nodes are selected, and dashed lines connect coalesced nodes.



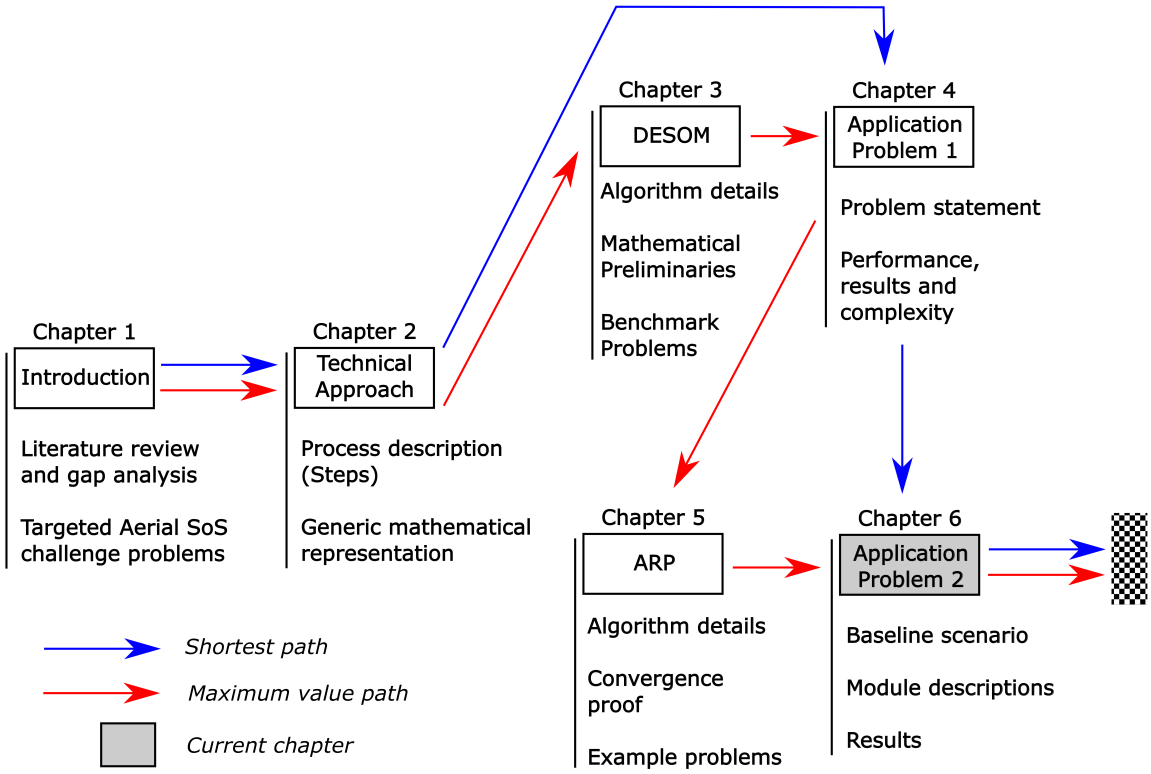
(a) Case b

(b) Case c

Figure 5.9.: Refer to cases in text : Case b) Final network after ARP selection. Case c) Network used when all candidate sensors are added a-priori. New sensors are colored gray.

5.5 Chapter Summary

An evolving counterpart of the standard minimum sum problem involving convex functions is presented. For the purpose of a formal proof of convergence, convex functions are utilized. However, demonstration problems also involved non-convex examples. We introduce a pivot phase in which new nodes (corresponding to new functions) are selected by first converting the overall optimization problem to a convex feasibility problem and then deriving the MFS. Proof of convergence of this Adaptive Random Projection (ARP) algorithm is provided. A relation for the upper bound of the rate of increase of the sequence $\{\gamma_k\}$ is an important area that is yet to be explored. We solve application problems which involve the minimization of the total compliance of structural topology, and robust source localization in the setting of evolving design spaces. In both problems, evolving the design space and solving using ARP allows us to improve the design by intelligently reusing previously obtained results. Comparisons were also made with alternate cases where nodes or agents selected by the ARP algorithm are added a-priori, and also when all candidate nodes are added at once. The algorithm is widely applicable, and may be important in problems where the subset of candidate nodes or agents to be added for an overall improvement is completely unknown. We now proceed to introduce and solve our final application problem.



We have now described and assembled the tools required to solve our final application problem. The problem solved takes the form of a tree of MDAO modules, and is made computationally tractable by a sequentially run parallel simulation with multiple workers on the Rossmann Super-computing cluster at Purdue University. A novel hybrid optimal control method used to determine trajectories to be flown at the SoS level is also introduced in this chapter. Other topics covered are an SoS CFD implementation to simulate the environment for multiple interacting aircraft, MDO architectures for individual level-specific modules and a holistic solution for an aircraft design problem involving commercial aircraft in the approach phase.

6. Application Problem 2

Nomenclature

BVP	Boundary value problem
DOF	Degrees of freedom
NLP	Non-linear Programming
MDO	Multidisciplinary Optimization
PCHIP	Piecewise Cubic Hermite Interpolating Polynomial
TPBVP	Two point boundary value problem

D	drag force magnitude, N	λ_{gam}	costate for relative flight path angle
L	lift force magnitude, N	λ_v	costate for velocity
L/D	lift to drag ratio	λ_x	costate for downrange distance
T	thrust force magnitude, N	λ_y	costate for crossrange distance
		λ_z	costate for altitude
g	acceleration due to gravity, m/s^2	σ	bank angle, rad
m	vehicle mass, kg	ψ	heading angle, rad
v	relative velocity magnitude, m/s		
t	time, s	p	PCHIP polynomial
x	downrange distance, m	ε	goal attainment factor
y	crossrange distance, m	o_i	objective i
z	altitude, m	g_i	goal i
		m	Number of objectives
α	angle of attack, rad		
γ	relative flight path angle, rad		

6.1 Introduction

One important aspect of SoS Engineering, often overlooked, is the role it plays in improving the design of individual systems. In this vein, our work has sought to exemplify such a synergistic approach for aircraft design. Also, aircraft design is a prime example of a SoS design problem with heterogeneous systems, interdisciplinary constraints, large number of interactions and high orders of complexity in design. Any increase in airport capacity must be sustainable and safe, and must abide by community noise restrictions. Recent research has revealed that aviation noise not only causes irritation to the members of a community that is close to an airport, but that continuous exposure to noise may have other long term health problems. [27] Apart from mitigating noise on the ground via insulation and the use of sound barriers, two strategies may be pursued:

1. Minimize aircraft (or System level) noise. This may include:
   (a) Engine noise
   (b) Airframe noise
   (c) Noise due to flaps, slats, extensions, speed brakes and landing gear
2. Minimize operational (or SoS level) noise.

Research has shown that aircraft flying in formation (an Aerial SoS) can improve aerodynamic efficiency. [153–155] Our application problem however does not intend to measure this improvement, but rather derives an operational procedure that these aircraft must follow for safe and efficient flight. These optimization problems that occur at multiple levels are described in further detail in the following sections. Note that further details are available as appendices in the supplementary material provided.

6.2 Baseline Scenario

Customers and stakeholders are often interested in holistic solutions that show improvement over the state-of-the-art or baseline solution. Furthermore, they are interested in de-

signs that span all levels of the hierarchy. In this section, we briefly describe the baseline solution corresponding to our aircraft design application problem. Aircraft are forced to fly inefficient, multi-staged landing procedures due to constraints related to separation assurance and safety, and for limitations posed by some legacy avionics modules. This is so that controllers may use the speed of an aircraft as a surrogate for distance. In the terminal area, controllers place aircraft at a 3 nautical mile (nm) distance from one another and ensure constant speed that is easy to monitor. We attempt to improve the CDA procedure Continuous Descent Approach (CDA) which has shown to have benefits in terms of time, fuel savings, and noise abatement. [156] The flight path angle at the intersection point (marked with a star in figure 6.1 is fixed at a constant 3 degree slope. Appendix C shows the airport map of the Austin Bergstrom airport. We design a minimum time trajectory to match the location, glide slope and heading at the intersection point (CHADE to runway 17R and DOFFS to runway 17L) before final approach.[1] Since the leading aircraft and the following aircraft in all our simulations are assumed to belong to the same weight category, a 2.5 nm separation is required on the final approach course.[2]

Recent projects launched by the FAA with regards to the conservative following distance used by controllers have reached major milestones in 2012 and 2013, with implementations of these procedures at several airports. The RECAT program [3] classifies aircraft according to its wingspan and ability to withstand wakes (in addition to the usual takeoff weight categorization). Implementing RECAT at Memphis has reportedly reduced arrival separations by 0.7 miles. A recent FAA order that is applicable to our problem is JO 7110.308, which determined that for aircraft following large or small leaders, using a parallel approach staggered by 1.5 nm was feasible and met safety mitigations [4]. The concept was successfully implemented at the San Francisco International Airport in September 2013.

[1] Airport maps were retrieved from www.aeroplanner.com (6/25/2015)

[2] Pilot and Air Traffic Controller Guide to Wake Turbulence, Technical Report, FAA. (URL - https://www.faa.gov/training_testing/training/media/wake/04SEC2.PDF)

[3] Wake Turbulence Recategorization program or RECAT - http://www.faa.gov/documentLibrary/media/Order/Final_Wake_Recat_Order.pdf

[4] FAA JO 7110.308 - http://www.faa.gov/documentLibrary/media/Order/JO%207110.308.pdf

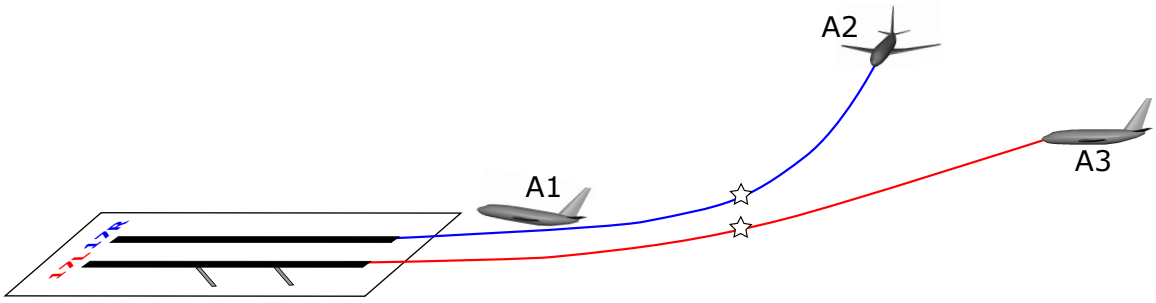


Figure 6.1.: Scenario involving three aircraft in the final approach phase. Note that the aircraft are not drawn to scale

Figure 6.1 shows a hypothetical scenario wherein three aircraft, A1, A2 and A3 are in their final stage of a standard, staggered approach towards the Austin Bergstrom airport with two parallel runways, 17L and 17R. In the situation shown in our hypothetical scenario, the order of landing is A1 followed by A3, followed by A2 (assume that in this perspective, the blue flight path segment to 17R is longer than the red segment to 17L). We also assume that departure operations do not affect the rate of arrival of aircraft into the airport (hereafter referred to capacity). Our aim is to safely increase the capacity of the airport system with respect to the baseline scenario by varying parameters that define the approach procedure (SoS level) while also meeting SoS level noise constraints, aircraft operating on these procedures (system level), and components that form aircraft (sub-system level). In the following sections, we describe key modules that reside in various levels of our solution framework. These modules are trajectory generation (section 6.3), noise analysis (section 6.4), wake turbulence analysis (section 6.5) and airfoil optimization (section 6.7). Once these modules are described, we then define the complex interactions between these modules (see section 6.8) that mimic real world engineering scenarios. Readers interested in the final results and implementation may direct their attention to the discussion in section 6.9.

6.3 Time-optimal trajectory generation

6.3.1 5-DOF Aircraft Model

The 5-DOF model used in this section is described by Eq.6.1 - Eq.6.6, where t is the time, x is the downrange, y is the crossrange, z is the altitude, v is the relative velocity magnitude, ψ is the heading angle, γ is the relative flight path angle, D is the drag force magnitude, T is the thrust force magnitude, L is the lift force magnitude, m is the mass of the aircraft, g is the acceleration due to gravity of Earth, and σ is the bank angle. [157] The effects of rotation of Earth have been ignored. The tangential and normal components of force on the vehicle are denoted by F_T and F_N respectively. During approach the velocity is assumed to be constant and hence acceleration as described in Eq.6.4 is zero. The variation of angle of attack, α then becomes a function of flight path angle, γ and is given by Eq.6.6.

$$\dot{x} = v\cos\gamma\cos\psi \tag{6.1}$$

$$\dot{y} = v\cos\gamma\sin\psi \tag{6.2}$$

$$\dot{z} = v\sin\gamma \tag{6.3}$$

$$\dot{v} = \frac{F_T}{m} - g\sin\gamma = 0 \tag{6.4}$$

$$\dot{\psi} = \frac{F_N\sin\sigma}{mv\cos\gamma} \tag{6.5}$$

$$\dot{\gamma} = \frac{F_N\cos\sigma}{mv} - \frac{g\cos\gamma}{v} \tag{6.6}$$

where

$$F_T = \mathrm{T}\cos\alpha - \mathrm{D}$$

$$F_N = \mathrm{T}\sin\alpha + \mathrm{L}$$

In the next section, we discuss the optimization problem formulated using the hybrid trajectory optimization technique. Details about equations and simplifications used in the indirect method can be found in Appendix A, and are closely related to the following discussion of our Hybrid trajectory optimization method.

6.3.2 Hybrid optimal control method for time optimal trajectory

Our objective is to use the best features of direct and indirect optimal control methods to obtain rapid, high quality solutions for minimum time optimal control problems. Our method converts trajectory optimization problems into equivalent parameter optimization problems which can be solved using existing Non-linear Programming (NLP) solvers. We add complexity to the standard parameter optimization problem (see [158]) by introducing additional costate variables and their corresponding time histories. In spite of introducing additional variables, higher quality solutions may be obtained more rapidly. The following discussion exploits this special structure. First we re-define the minimum time optimal control problem as a minimize final point error problem. The problem structure is changed to satisfy end-point boundary conditions, and we acknowledge that the co-state dynamics must also be modified. The original objective (minimum time) is retained in a multi-objective strategy to be discussed shortly. The redefined problem is as follows:

$$J = \Phi(t_f, x(t_f)) + \int_{t_0}^{t_f} L(t, x(t), u(t))dt \tag{6.7}$$

$$\dot{x} = f(t, x(t), u(t)) \tag{6.8}$$

$$x(t_0) = x_0 \tag{6.9}$$

$$\phi(t_f, x(t_f)) = ||x(t_f) - x^*_{t_f}||^2 \tag{6.10}$$

where $x^*_{t_f}$ contains the optimal values of the state variables at the end point that need to be achieved. Note that when certain state parameters are free in the original problem, components of the norm associated with these free parameters can be excluded. The path cost L is now 0 instead of 1. As in Indirect methods, we use the necessary conditions of optimality given Euler-Lagrange equations (see Eq.A.20-A.21 in Appendix A). λ in the following equations is an n-dimensional costate vector, and H is the Hamiltonian. We then follow the indirect methods to derive the costate dynamics and optimal control as in Eq.A.22-A.29 (in Appendix A), with a minor deviation in the equations that represent the costates λ_x, λ_y and λ_z:

$$\dot{\lambda_x} = 2 \cdot (x(t_f) - x^*_{t_f}) \tag{6.11}$$

$$\dot{\lambda_y} = 2 \cdot (y(t_f) - y^*_{t_f}) \tag{6.12}$$

$$\dot{\lambda_z} = 2 \cdot (z(t_f) - z^*_{t_f}) \tag{6.13}$$

At this juncture, we deviate from the classic indirect methods. State and costate time histories are represented as Piecewise Cubic Hermite Interpolating Polynomials (PCHIP) as a function of the normalized time that ranges from 0 to 1. Note that in practice, the dynamic equations must be scaled by t_f to maintain equivalence. PCHIPs maintain the monotonicity of the original data while also preserving the interpolation, which is reflective of the time histories of the physical quantities we wish to represent using these interpolants. [159] Furthermore, these curves (that represent time histories of variables) can be easily differentiated and integrated during the course of solving if necessary. This allows us to convert our original TPBVP, to an End-point problem. PCHIPS representing the states, costates, and controls (and differentials and integrals of these time histories) are now controlled using sets of parameters specific to each of these quantities with specified starting points. As mentioned earlier, components of the states or costates which are fixed and free are either excluded or included in the set of optimization design variables of the NLP respectively.

In our problem, since acceleration $\dot{v} = 0$, our velocity along the trajectory is constant but unknown and is therefore introduced as another parameter or design variable in the optimization process. Since x, y and z are purely functions of v, ψ and γ, we only use PCHIPs for this subset of states. Lastly, we define parameters to represent the PCHIPs of $\dot{\lambda_\psi}$ and $\dot{\lambda_\gamma}$. Note that the values of all costates at t_f are obtained using the transversality condition as 0 with $u*$ as control. This allows us to make appropriate adjustments in the curve obtained during integration of the costate equations. In summary, we convert the indirect optimal control problem into a nine-parameter optimization problem involving two intermediate control points in each of $\psi(t)$ and $\gamma(t)$, three control points of λ_γ, value of velocity v along the trajectory, and the value of t_f. The value of t_f is useful in scaling of the dynamics of state variables to represent the same in the interval $[0, 1]$, and also as part

Variable	ψ_1	ψ_2	γ_1	γ_2	$\lambda_{psi,1}$	$\lambda_{psi,2}$	$\lambda_{psi,3}$	v	t_f
Lower Bound (y_{lb})	-pi/2	-pi/2	-pi/6	-pi/6	-50	-50	-50	200	30
Upper Bound (y_{ub})	pi/2	pi/2	pi/6	pi/6	50	50	50	250	60

Table 6.1: Values of lower and upper bounds of variables used in the hybrid optimization problem

of the objective function which helps us relate back to our original interest - time optimal trajectories. We pose this NLP problem as a constrained multi-objective goal attainment problem involving two objectives o_1 and o_2:

$$\begin{aligned} o_1 &: (x(t_f) - x^*_{t_f})^2 + (y(t_f) - y^*_{t_f})^2 + (z(t_f) - z^*_{t_f})^2 + t_f \\ o_2 &: \max(\text{abs}(H(t))) \end{aligned} \tag{6.14}$$

While o_1 drives the appropriate end states to the desired end point while minimizing time, o_2 drives the value of H to zero at all points of time t, which is desired in our problem. Since the final state constraints are expected to be satisfied, our problem is equivalent to the vector optimization problem involving minimizing t_f and driving H towards zero. The 'goals' we wish to drive these objectives are thus $g_{t_f} = 0$ and $g_H = 0$. It is easy to see how one can modify these goals to satisfy any other optimal control problem. In our formulation of the problem, the constraints are in the form of lower and upper bounds of the variables that the parameters represent to force realistic values. These constraints are listed in table 6.1.

The multi-objective goal attainment problem tries to find the parameters y that minimize the maximum of the objectives o_i, given a set of weights w_i:

$$\min \max \frac{o_i(y) - g_i}{w_i} \tag{6.15}$$

by converting the problem into a simpler form involving a single parameter $\varepsilon \in \mathbb{R}$:

$$\min_{y} \quad \varepsilon \tag{6.16}$$

$$\text{S.To.} \quad o_i(y) - w_i\varepsilon \leq g_i, \ \ \forall i = 1, \ldots, m \tag{6.17}$$

$$y_{lb} \leq y \leq y_{ub} \tag{6.18}$$

$$\tag{6.19}$$

In our current problem, $m = 2$. Our implementation involves the use of *fgoalattain* function in the optimization toolbox of MATLAB, which itself is based on the works by Gembicki [160], Han [161] and Powell [162] and has proved to be globally convergent for certain classes of NLP. Readers are reminded that due to the method of calculation of trajectories (not involving classic dynamic propagation and root solving), the error in the final trajectories involve 1) deviation from the optimal answer, and 2) deviation of state and co-state dynamics. The optimizer used must be able to drive H and the boundary constraints to zero (practically, with some tolerance) for all time. Since it is difficult to find an optimizer that achieves this for *all* problems, user discretion and tuning is necessary.[5]

6.4 Noise Analysis

6.4.1 System Level - Minimizing Airframe Noise

The bottom-up design of aerospace vehicles is a process that involves multidisciplinary efforts in a vast design space. At the very core of this sub-problem lies the USAF Stability and Control Digital DATCOM, which provides a systematic summary of methods for estimating stability and control characteristics. An open source version of the software, Digital DATCOM was used in the tool, as will be discussed later. Antoine and Kroo used MDO to determine the extent to which noise can be traded against other performance measures. [163] This work showed that, of the different figures of merit that were optimized

[5] A future version of this algorithm will involve the use of the Covector Mapping Theorem for obtaining the co-state trajectories from the Lagrange multipliers of the optimizer used (**a feature of some direct methods**), and also propagating dynamic equations throughout the complete time history to obtain physical/realistic trajectories at all intermediate steps (**a feature of indirect methods**

(takeoff weight, operating cost, noise, nitrous oxide emissions, and fuel burn), optimization for noise required the greatest concessions in the other potential objectives. The engine, airframe, and the interference between the engine and airframe are the main sources of the aircraft noise. Airframe noise represents a lower bound or minimum noise generated by the aircraft, and is a large percentage of the total noise during approach.

The Authors' evolutionary algorithm, Differential Evolution with Self-Organizing Maps (DESOM) was used as a top-level optimizer. [45] The algorithm uses the neural network variant, SOM to accelerate convergence towards an optimum design with reduced function evaluations. This is required since the objective function (see discussion below) is multimodal, non-linear and is computed as a result of a simulation in the black-box function, DATCOM which consumes approximately 10 seconds of processing time. The objective function is a modified version of the Lockard and Lilley [164] noise metric as presented by Leifsson [165] a form that is used to approximate the far-field noise intensity (I) of a clean-wing at high lift (see equation below):

$$f(x) = I = \frac{1.7}{2\pi^3} \frac{v_\infty M_\infty^2 W}{C_L H^2} \left(1 + \frac{C_L^2}{4}\right) \tag{6.20}$$

for level flight at a particular altitude (H) and velocity (v_∞) that corresponds to Mach number (M_∞) and coefficient of lift (C_L). The equation only models clean-wing noise, and does not capture the effects of engine noise or noise due to high lift devices like slats and flaps. It is important to minimize noise intensity of the clean configuration at approach since it acts as the lower bound of all noise sources including flaps, slats, engine noise and landing gear noise (see Lockard and Lilley [164] for a discussion of clean configuration noise in the approach phase). Since DATCOM was used as a black box function, several design variables were required to define a single aircraft (in 3D). The table below defines the "namelists" (groups of design variables) used. Note that in this study, the design variable space is reduced to have 42 dimensions by keeping the fuselage fixed (accounting for an additional 9 variables) to that of the baseline design, a Boeing 737.

DATCOM input files were supplied by a custom MATLAB wrapper, which was then executed to provide data used by the objective function and constraints. The 35 discipline-

Name-list	Significance	Variables	Number
Synths	Configuration of wing, stabilizers and Fuselage	L, xw, xh, zh, zw, xv, aliw, xvf, zvf, bnose, btail	11
Wgplnf	Wing planform parameters	chr, chrdt, type, chrdbp, sspan, sspanop, savsi, sspanne, savso, dhdadi, dhado	12
Vtplnf	Vertical stabilizer parameters	chr, chrdt, sspan, sspanne, twista, savsi, savso, dhdadi	8
Htplnf	Horizonal stabilizer parameters	chr, chrdt, sspan, sspanne, twista, savsi, savso, dhdadi	8
Wsec	Aifoil selection - wing and stabilizers	w, h, v	3
Fuse	*Fuselage shape parameters*	*ay1, ay2, ay3, by1, by2, by3, cy1, cy2, cy3*	*9*
Total			42 (*51*)

Table 6.2: Design variables used in the aircraft level problem (42 of the possible 51 design variables are used here). The Name-list column uses group names as seen in the DATCOM input file.

specific constraints were added to the objective function using the linear form as shown in Equation 6.21. ϕ is the pseudo-objective function, and r_p is a user specified penalty multiplier (here we used a constant value of 20 for rp). We recognize that the final optimum values and design vectors are sensitive to the value of rp, but a comprehensive sweep of tests to determine the best rp value is out of the scope of this work and will be discussed in a later publication.

$$\phi(x) = f(x) + \sum_{i=1}^{35} g_i(x) \tag{6.21}$$

The 35 constraints used may be broadly classified as configuration-based and performance based. Configuration based constraints ensure that the aircraft maintains sensible proportions. The configuration constraints appear as linear, upper and lower limits. The constraints $g28$ to $g35$ are non-linear performance constraints. Although some of these constraints may appear linear, their relation with the design variables are highly non-linear. These values are derived from the output of a full simulation at a specific Mach Number M, and a range of angle-of-attack (α) values. The list of mathematical constraints used, along with their significance are shown in Appendix E. At this level, our implementation corresponds to the popular MDO architecture, Individual Discipline Feasible (IDF) with a top level optimizer (DE-SOM2) that enforces coupling between all disciplines by using a common global design. Modules containing equations related to weights and balance, configuration, operation (landing, tail scrape angle etc.), aerodynamics and stability use the DATCOM analysis output in the IDF framework.

6.4.2 SoS Level - Monitoring Noise Exposure Contours

Several innovations at the SoS (or operational) level have improved noise levels, but have worsened community perception of noise. A combination of on-site noise monitoring and development of accurate noise models have helped reduce noise levels over the years. Alternate forms of the equivalent noise exposure metric (L_{eq}) have been developed and used by the Federal Aviation Administration (FAA) through their Integrated Noise Model

(INM), and the Civil Aviation Administration (CAA) through their noise model ANCON (Aircraft Noise Contour Model 1). Both INM and ANCON have significant similarities and relate noise sources to 4D trajectories of the aircraft. The L_{eq} value measures "annoyance", which can be related to community impact. Readers interested in other short term noise exposure metrics are referred to the technical manuals of INM and ANCON. [166–168] An appropriate operational or SoS-level measure of noise is the area enclosed by the 57 dB L_{eq} noise contour. For example, despite the growth in number of flights arriving at Heathrow, the noise contour area has decreased by almost 90% since the early 1970s, with little or no improvement since 2000. [27]

We use CAA's ANCON noise model at the SoS level to analyze and monitor the noise contour area. It is typical to use a set of L_{eq} noise contours from 51 dB to 72 dB in steps of 3 dB to judge low, moderate and high annoyance. Note that the 57 dB L_{eq} contour is the current "accepted representation of the onset of annoyance". Given an aircraft trajectory, an SoS-level requirement is to decrease the area of a noise contour with respect to baseline aircraft traveling on baseline trajectories. The resolution (distance between virtual monitoring sites) is set to 10 m.

Our noise contour calculation is an implementation of CAA's ANCON detailed in the official technical report. [169] At the core of the L_{eq} model is the calculation of a sound exposure level assuming that the aircraft generates noise as a constant source along an infinite path at constant velocity, flight path angle and heading. The calculation also depends on a corresponding database of measured maximum noise (*Lmax*) levels. In our work we obtain this value of *Lmax* from the recorded average arrival noise levels for Boeing 767-300 with PW4000 engines. [168] [6] The model then accounts for various adjustments to the base noise level using atmospheric noise dissipation, Noise and Number Index (NNI) attenuation, directivity patterns, lateral attenuation, wing-mounted engine installation effects, and acoustic impedance adjustment. However, our implementation does not assume effects of

[6] Noise data for the B737 with the new CFM56-3 engines were not publicly available at the time of writing this thesis.

terrain or line of sight blockage, and no thrust reversal adjustment post landing (which are all additional attenuation factors used by FAA's INM and CAA's ANCON).

6.5 Aircraft Wake Turbulence Analysis

6.5.1 System Level - Augmented Betz Method

At the System Level, we first compute the wake vortex of the aircraft traveling at a particular Mach number at a fixed number of spans behind the aircraft. At this level, we choose to use an analytical model over more complicated models like the Navier Stokes due to the need to balance available resources with other modules in other levels. Ning *et al.* suggests the use of the Betz model for inviscid wake roll-up over other empirical models such as the Rankine model and the Lamb-Oseen model since it does not assume a form for the vorticity distribution but rather calculates it based on the lift distribution (which is obtained from the DATCOM software in our case). [170] The only assumptions made are that 1) the vortices generated by the lifting body are asymmetrical; 2) negligible interaction between two active vortices; and 3) The core size does not grow and is about 1-2 % of the wingspan. Also, our use of the Betz model can be justified by pointing to the surprisingly accurate predictions made in comparison to experimental tests. [170, 171] The theory relates the wing loading Γ and span-wise station y to give the circulation in the rolled-up vortex shed (Γ') as a function of radial distance from the center of the asymmetrical vortex.

Donaldson outlines the procedure to calculate the center and strengths of multiple vortices shed across the wing. [171] This procedure is repeated here briefly for completeness: 1) Obtain the circulation over the wing (Γ), 2) Calculate the distribution of the shed vorticity $d\Gamma/dy$, 3) Calculate the number and location of the minima of $|d\Gamma/dy|$ (these are the regions of where vortices are shed from, which will include the tip vortex), 4) Calculate the positions of maxima of the distribution, $|d\Gamma/dy|$ (these are locations of the center of the vortex), 5) Given a region in (3) that corresponds to a center or maxima (ym) in (4), select two points $y1$ and $y2$ equidistant from the center ym. The circulation of the shed vortex at

a radial distance r from ym is equal to $\Gamma(y1) - \Gamma(y2)$. Finally, wake roll-up may be calculated by displacing these centers using the effect of upwind vortices and wind/traveling velocity. A typical Rankine vortex representation is used, where the maximum tangential velocity is limited to $V_{\theta,max} = 886 \cdot t^{-1/2}$, at a core radius $rc = \Gamma_{max}/(2\pi V_{\theta,max})$, assuming the wake is fully developed after t seconds (here, we consider t=10 seconds, and Γ_{max} is 6000 ft^2/s). [172–174] [7] The velocity distribution in a vortex shed by the wing according to the Rankine model is given as:

$$V_\theta = \begin{cases} \frac{\Gamma}{2\pi rc}\frac{r}{rc} & \text{, if } 0 < r < rc \\ \frac{\Gamma}{2\pi r} & \text{, if } r \geq rc \end{cases}$$

A 'snapshot' of the velocity distribution at $t = 10$ seconds is used in the CFD atmosphere of the SoS level to propagate wakes to the entire field. Given the relative location of a leading aircraft, the following aircraft must travel through consecutive sections of the wake while maintaining a safe distance [154, 155]

6.5.2 SoS Level - Computational Fluid Dynamics Simulation Environment

Each aircraft in the SoS simulation environment is represented as a 5 DoF point mass traveling on a time-optimal trajectory. Velocity distributions obtained by the Augmented Betz method are then propagated or developed in the SoS simulation environment, which is essentially a CFD box. Note also that the trajectories are being monitored to check if the aircraft operations satisfy noise constraints. The incompressible, isothermal Navier-Stokes (NS) equation is used to simulate the aircraft traveling through the CFD box, which is assumed to contain air (considered to be Newtonian). A form of the NS equation known as the Pressure Poisson equation (PPE, derived from the momentum equation) takes advantage of the weak coupling between pressure and velocity given boundary conditions. [175] Equivalence of the PPE to NS is demonstrated by Cornthwaite. [176] While [176] uses an Galerkin Finite element Method, we use an explicit time-stepping method to recover pressure from

[7] These empirical relations are derived after extensive tests conducted on B-737. [172, 174]

velocity distributions (which are discretized using a central scheme), and then update the pressure field through PPE. The implementation is similar to the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) algorithm which can be found in [177]. Note that for numerical stability, we follow the recommendations of the CFL (Courant Friedrichs Lewy) condition threshold. More details may be obtained from [175–178].

6.5.3 Trajectory adjustment based on CFD environment

The calculated time optimal trajectory does not take into account other aircraft in the airspace as is. An aircraft may follow another aircraft in the arrival airspace by estimating a safe following distance rather than relying on empirical tables. In our implementation, the original trajectory can be modified based on the existing flow-field of the SoS CFD environment. Numerical computations and experiments have shown that aircraft traveling in the upwash generated by leading aircraft can also experience drag and fuel saving benefits. [153,154,170] We modify the original trajectory by first identifying a new path through regions close to sections along the trajectory with maximum drag benefit. Then we add the root mean square (RMS) distance from the original time-optimal trajectory as an additional objective function in our Hybrid optimal control method. Figure 6.2 illustrates this modification of the original trajectory (in blue) to a new, improved one (in red) with drag benefits. The “drag benefit” contour sections (as they are popularly referred to in literature) shown along the trajectory are relevant regions of interest to help modify the original trajectory (red regions have higher benefit in this diagram). Obviously, the modified trajectory may not be time-optimal.

Note that although our main goal is to actively judge the distance at which an aircraft must follow a leading aircraft rather than rely on thumb-rules and tables, our ‘drag benefit trajectory’ may allow following aircraft to operate at a lower throttle level, therefore also saving fuel and reducing noise. These are only by-products of our implementation.

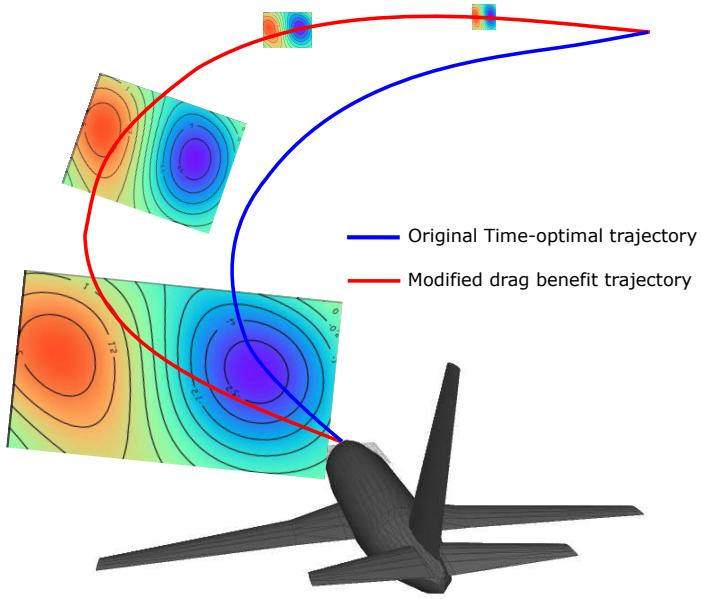


Figure 6.2.: Original time-optimal trajectory is adjusted to obtain a new trajectory with drag benefit. The popularly used "drag benefit" trajectory is used to ascertain a safe following distance dynamically

6.6 Aircraft Engine Analysis

A custom *MATLAB* Simulink model of a conventional turbojet with single compressor and turbine stages was constructed using the Aerospace Toolbox and corresponding physical systems library.[8] We wish to use the Engine analysis module to check if a constant (required) thrust can be maintained by controlling the mass flow rate in the presence of uniform internal system noise and changing altitude (and therefore, changing ambient atmospheric parameters) . The internal white-noise introduces fluctuations in the thrust output of the Engine, which in turn affects the specific fuel consumption, fuel-to-air ratio, and fuel flow rate and output dB level of sound. Although the amplitude of the noise has a maximum value equal to 10% of the required thrust, fluctuations in output thrust may prevent the aircraft from traveling at a (close-to) constant velocity along the trajectory as

[8]A basic version of a turbojet is available in the Aerospace toolbox of Simulink called *sscturbojet*. A later version of this work will involve modeling of a High Bypass Turbofan engine, which is a more accurate representation of the engine used in commercial aircraft

required by our 5DoF model during approach. Only the core turbojet is modeled here for simplicity. Further studies will include a more realistic turbofan engine model.

At the intake, Mach number and ambient temperature (assumed constant) are used to calculate the total pressure and inlet temperature. This is calculated using equation 6.22, where subscripts i and a stand for inlet and ambient conditions respectively.

$$\begin{aligned} T_i &= (1+(\gamma-1)/2M^2)\cdot T_a \\ P_i &= P\cdot(1+(\gamma-1)/2M^2)^{(\gamma/(\gamma-1))} \end{aligned} \tag{6.22}$$

Next, an adiabatic process at the compressor (see Equation 6.23) is used to increase the pressure of the intake are 8-fold (fixed 8:1 pressure ratio). In the equations below, PR_C, TR_C and η stand for compressor pressure ratio, temperature ratio and efficiency respectively.

$$\begin{aligned} P_c &= PR_C\cdot a.p \\ T &= T_i\cdot(1+(1/\eta)\cdot((PR_C)^{((\gamma-1)/\gamma)-1)}) \\ TR_C &= (1+1/\eta\cdot((CPR)^{((\gamma-1)/\gamma)-1)}) \\ T_c &= T\cdot TR_C \end{aligned} \tag{6.23}$$

The burner or combustor adds heat at a constant pressure, and therefore increases the temperature almost instantaneously (to T_B). This air at a high pressure and temperature is passed through the turbine. the energy balance between the turbine and compressor is used to calculate the outlet pressure and temperature. Notice the use of bypass air at temperature equal to the inlet temperature.

$$\begin{aligned} T_T &= T_B - T_i\cdot(TR_C-1) \\ P_C &= P_C\cdot(1-1/\eta(1-T_B/T_a))^{(\gamma/(\gamma-1))} \end{aligned} \tag{6.24}$$

The nozzle then further accelerates the flow by decreasing the pressure to ambient pressure. The *MATLAB* model uses empirical tables and tabulated logs for this step (unavailable to public).

A custom engine controller module is used to continuously modify the reference mass flow rate that is required to increase or decrease thrust by the appropriate amount to attain the required thrust at the current operating conditions and time during the trajectory. *MATLAB* control systems toolbox is used to automatically tune the PID controller for every trajectory that is flown. A standard compensator formula for the PID controller is used:

$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}$$

where P, I and D are the proportional, integral and derivative gains, and N is the filter coefficient. Saturation blocks are used to limit the maximum producible thrust and maximum achievable mass flow rate. The *MATLAB* Simulink model used is shown in figure 6.5 (see Appendix B).

6.7 Airfoil Aero-structural Optimization

Although aircraft modeled in the System level optimization problems have the option to choose from a library of over 60 airfoils for use in constructing its wings, vertical stabilizers and horizontal stabilizers, targets or requirements from the upper levels (System- and SoS-levels) may drive the airfoil design towards a particular region in the design space. Airfoil section characteristics like efficiency, zero-lift drag and pressure distribution have a direct impact on wings constructed using these airfoils. Apart from aerodynamic impacts, we also use airfoils to form ribs, which are important structural members which are subject to stress and fatigue. At the sub-system level, we may only attempt to find an optimum aero-structural configuration of *normalized* airfoil, i.e, an airfoil with a chord length of one unit since wing planform shapes are a result of the upper level aircraft optimization problem. Our implementation is a single point optimization run, and is multi-disciplinary in nature. We could however, use a multi-point strategy to add value to the results obtained. We now describe a combination of evolutionary optimization of airfoil shapes for maximum aerodynamic efficiency (C_L/C_D), along with topology optimization to define an internal structure while minimizing compliance. To coordinate these disparate optimization problems, we

use the MDO architecture called Enhanced Collaborative Optimization (ECO). [179, 180] This architecture choice suits our problem set up since the top level optimizer only strives to achieve compatibility between the disciplinary sub-spaces (here, this corresponds to the airfoil boundary or shape), while the discipline-specific optimization problems maintain independence (See Appendix F for the Extended Design Structure Matrix or XDSM for our ECO implementation).

6.7.1 Aerodynamic Optimization Module

The airfoil optimization problem used in 3.5 is repeated at the Sub-system level. However, unlike application problem 1, the sub-system level now involves multi-fidelity aero-structural optimization instead. In addition to this, the structural optimization module involves an evolving design space of type 2 (using ARP), and also a final parameter sensitivity analysis. The aerodynamic optimization module is re-described here in brief for completeness. The Class/Shape Transformation (CST) technique introduced by Kulfan *et al.* controls the shape of the airfoil. Therefore, our design variable x has six dimensions. The design space considered along with some sample airfoils is shown in Fig. 6.3.

In the trials conducted, a 'function evaluation' involves writing an input file to the *Xfoil* program, analyzing the airfoil that airfoil at a Reynolds Number (Re) of 10^6 and a Mach number (M) of 0.2 for a range of angle of attacks ($\alpha = -5$ to 25). The maximum $\frac{Cl}{Cd}$ (regardless of the corresponding α) is reported as the objective function value. In case *Xfoil* is not able to converge (usually due to an unconventional airfoil shape), we attribute that design point with a random, large, positive number so that the same design point is

avoided in the next generation. Our bounded, single objective optimization problem can thus be formulated in Eq. 6.25:

$$\begin{aligned} \text{Minimize} \quad & -\frac{Cl}{Cd} \\ \text{Subject To} \quad & LB \leq x \leq UB \\ \text{Where} \quad & LB = [0.05\ 0.05\ 0.05\ \text{-}1.00\ \text{-}1.00\ \text{-}1.00]^\top \\ & UB = [1.00\ 1.00\ 1.00\ \text{-}0.05\ 0.00\ 0.00]^\top \end{aligned} \tag{6.25}$$

Note that the problem is posed as a minimization problem with the appropriate sign change for the objective function. A graphical representation of the airfoil design space (which are formed by the six parameters in x) is shown in Fig. 6.3. A variety of airfoils with upper surfaces in the red region and lower surfaces in the blue region can be formed (a sample feasible airfoil is also shown).

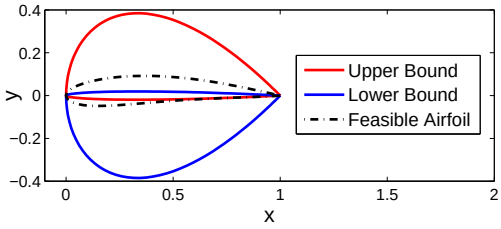


Figure 6.3.: Graphical representation of the airfoils formed with all six design variables at the lower bound (blue), and at upper bound (red). A feasible airfoil (one that lies between the upper and lower bounds) is also shown (black dashed lines). Distances are non-dimensionalized.

6.7.2 Multi-fidelity Analysis

Airfoil analysis may be performed at two levels of fidelity. In our chapter on Application problem 1 (Chapter 4), we loosely defined fidelity according to an integer 'level' along with the average time taken to complete a particular analysis run. [181] Our airfoil optimization in this application problem can take advantage of two levels of fidelity - inviscid analysis and full viscous analysis. The inviscid formulation used by Xfoil is a linear-vorticity stream function panel method, and the viscous model uses boundary layers and wakes described by a two-equation lagged dissipation integral formulation. [182, 183] 500 iterations are used for convergence in the case of viscous analysis (average of 6.012 seconds per run) as compared to 100 for inviscid

analysis (an average of 2.479 seconds per run). Any process that takes more than 10 CPU seconds is killed externally. Note that although validations were made with respect to experimental runs, standard error was not reported. This precludes us from using the original VoI formulation.

6.7.3 Structural Optimization Module

The purpose of topology optimization is to find the optimum layout of a structure given the domain (here, the airfoil boundary), loads and distribution of material. The Messerschmitt-Bolkow-Blohm (MBB) beam problem is a classical problem in topology optimization literature and is modified to form the structural module of the airfoil aero-structural MDO problem. [137] The objective of this sub-problem is to minimize compliance with the constraint on the amount of material inside the airfoil boundary (therefore, minimizing mass is not the objective). Formally stated, the problem we wish to solve is:

$$\begin{aligned} \min_x c(x) &= U^T K U = \sum_{i=1}^{N} E_i(x_i) \cdot (u_i)^T k u_i \\ \text{S.To. } & V(x)/V_0 = v_{frac} \\ & KU = F \\ & 0 \leq x \leq 1 \end{aligned} \tag{6.26}$$

where $c(x)$ is the compliance of the airfoil structure, U, F and K are the global displacement vector, force vector and stiffness matrix for an element with unit Young's modulus respectively. $V(x)$ and V_0 are the material volume and domain volume, and v_{frac} is the volume fraction. For a given domain size, v_{frac} is fixed. As in section 5.4.1 this value is adjusted to maintain the total amount of material used as we evolve the airfoil. The objective function is written as a sum of compliances of individual elements with u_i being the displacement vector of each element, and k being the element specific stiffness matrix. It is assumed that the Young's modulus of each element is obtained as a function of densities x_i according to Equation 6.27 shown below.

$$E_i(x_i) = E_{min} + x_i^p (E_0 - E_{min}) \tag{6.27}$$

The values of E_{min}, E_0 and p used here are $1e^{-9}$, 0 and 3 respectively. Guiles' work on multi-complexity structural topology optimization is similar to our work in that the outer domain size decides the number of elements, and therefore the number of variables. [184] An airfoil with larger internal area would need more number of descriptive design variables. While the domain used by Guiles is the entire wing, discretized by shear webs, spar caps and skin elements, we discretize the airfoil that would appear as ribs at three equidistant locations (root, tip, and mid-point) on the wing with density elements. The phenomenon of checkerboxing in resulting solutions here may be prevented in the same way as reported in Guiles [184], that is, by prescribing the perimeter of the domain beforehand. More information about the optimality criteria, sensitivity studies and alternative solution methodologies may be obtained from a wide variety of existing literature. [137–142] It is important to note that the material considered here is isotropic and linearly elastic.

A significantly modified version of the 88 line topology optimization code by Sigmund is used to find the optimum internal structure of an airfoil with the minimum compliance. The above optimization problem is solved using an *Optimality Criteria method* through a proven heuristic updating scheme. [139] The number of variables involved ($|x_i|$) is determined by the shape of the outer boundary (here, the airfoil boundary) and the resolution (chosen value is $1/60$ across all our simulations). Given this resolution, the maximum number of variables for the structural optimization problem is 1200, which corresponds to a rectangular airfoil in a 60×20 space. In reality, the number of variables is closer to about 800 for airfoil shapes.

We gauge the robustness of the final design by performing sensitivity analysis as a post-optimization step. This is distinctly different from Robust Optimization wherein parameter or design variable uncertainty is incorporated into the optimization loop itself. [185] During optimization runs, the point of application of the unit force is at 80% ($\theta = 0.8$) of the chord length. For sensitivity analysis, we use the final optimum airfoil generated by the

aero-structural optimization problem and test it against a uniformly distributed set centered around $\theta = 0.8$ with a resolution of 0.01 and a sample size of 100.

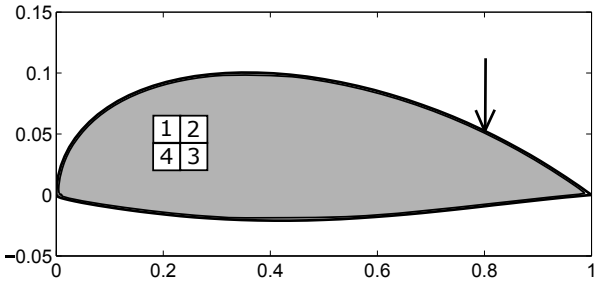


Figure 6.4.: Airfoil structural optimization problem definition in an evolving design space to be solved using ARP. The arrow represents the unit force applied at 80% chord

We employ the use of Adaptive Random Projection (ARP) to solve a slightly modified problem shown in figure 6.4. Our aim is to use ARP to find an airfoil that has a superior $f*$ value than the final optimal airfoil, given that a hole in the shape of a square spar is to be drilled through. The spar cross section covers the area that is responsible for 16 variables. These 16 additional variables are controlled by 4 agents (labeled in the figure). ARP is used to decide which agents (and therefore which variables), and what combination of agents must be selected to be added to the base design for in improvement in the previously calculated optimum value, $f*$.

6.8 Summary of Linkages Between Optimization Levels

As we can see from the above description (sections 6.3 to 6.7), SoS optimization problems are often complex and detailed in nature. This is to ensure realistic solutions that may be used in practical scenarios. Also, stringent customer requirements and market demand characteristics often drive the need to build multi-fidelity, multi-disciplinary models to ensure high levels of confidence with respect to the final solution offered. Given the complex models used in this example problem, our SoS optimization framework simplifies the problem solving process by placing these modules in a tree of MDO problems. When

posed as a tree of MDO problems, the interactions between modules and optimization processes become very important in ensuring tractability and solution quality. Thus, process and data flow between the MDO problems and analysis modules belonging to different levels in our three-level SoS optimization problem are critical to the overall functioning of the framework, as well as the solution itself.

Figure 6.5 describes the inter-level and intra-level interactions between modules used to assemble the SoS optimization problem. The ECO MDO framework is used to link the optimization modules contained in airfoil aerodynamic and structural analysis. Note that the airfoil aerodynamic optimization is a multi-fidelity analysis block. ARP is used to find if an airfoil structural compliance may be decreased by evolving the design space. This airfoil is then fed to the IDF framework at the system level for minimizing airframe noise of an aircraft analyzed using DATCOM. The lift distribution over the wing is used to derive the multiple-vortex system in the downwash of the wing after 10 seconds of wake development. This wake snapshot is used by the SoS CFD environment an propagated in time. Using properties of available or newly generated aircraft, time optimal trajectories are calculated while monitoring noise levels and safe separation distances given the wakes generated by leading aircraft.

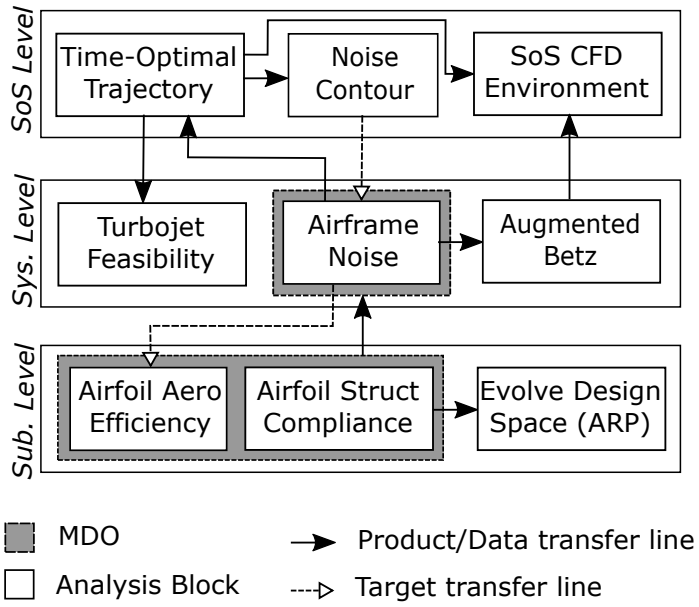


Figure 6.5.: Summary of linkages in the given SoS optimization problem

6.9 Results and Discussion

Simulations were run using several MATLAB Linux instances as jobs submitted to a cluster computing system at Purdue (Rossmann Cluster). A maximum of 3 compute nodes were used at any given point of time - Each node is equipped with a 2.1 GHz 12-core AMD 6172 processor and 192 GB of RAM and a 10 GigE Interconnect. The jobs were not data intensive, but involved the use of custom Python and MATLAB codes, commercially and freely available software and data transfer between the levels of the SoS optimization problem. As per directions from the capability flow-down concept used in the framework, we first attempt to find a solution in the top-most (SoS) level by improving approach procedures for existing systems that are composed of existing sub-systems. If no improvement can be made with changes at the top level alone, instructions to design better systems and sub-systems cascade downward, with improved products from each level flowing upwards. In this specific aircraft design example, we see that the entire three level hierarchy needs to be active in creating new and improved products or designs at each level. We will now present level-specific results.

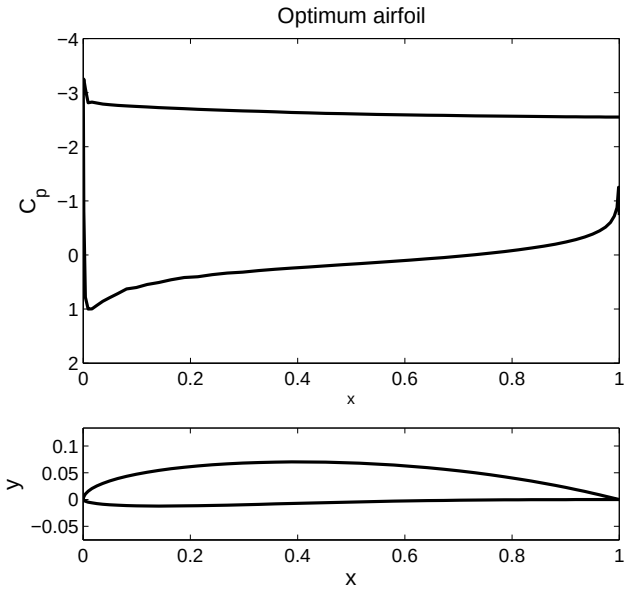


Figure 6.6.: Optimum airfoil with maximum $C_L/C_D = 61.03$

6.9.1 Sub-system Optimization

At the sub-system level, airfoil optimization and Engine control analysis were performed. The final airfoil generated with maximum C_L/C_D is shown in figure 6.6 with its corresponding plot of coefficient of pressure as analyzed in the high fidelity module of Xfoil. The airfoil level is triggered by the System level since the baseline aircraft does not satisfy the original requirements of posed (more details in the System optimization section).

The convergence history of DE-SOM2 (shown in figure 6.7) includes plots of the convergence to the final value of maximum C_L/C_D as well as the values of *pswitch* which decides roughly the percentage of function evaluations that is carried out using DE versus DE-SOM2.

The structural optimization module generates an internal structure for a given airfoil boundary throughout the convergence history. Structural optimization is conducted once every 20 generations of airfoil aerodynamic optimization. Readers are reminded that a point load is applied in the downward direction at 80% of the chord. Here, we extend

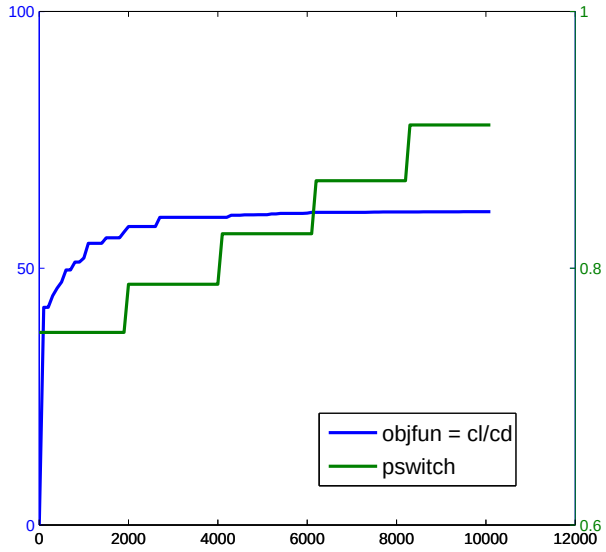


Figure 6.7.: DE-SOM2 convergence history including objective function and *pswitch* values

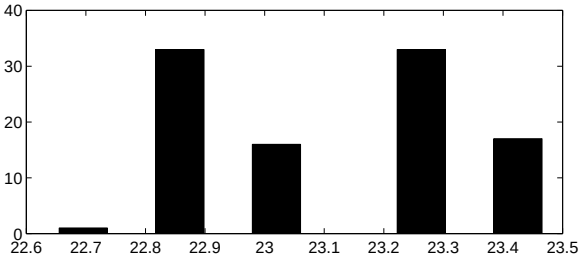


Figure 6.8.: Distribution of objective function values with respect to a uniformly varying point of application about the 80% chord location.

this simple, single point analysis by visualizing the histogram of the objective function corresponding to the distribution of load application points (uniformly distributed samples of positions from 75 to 85 % of the chord) in figure 6.8. A bi-modal distribution is observed with an overall objective variance of 0.051. As a post-optimization step, this sensitivity analysis may be used by designers to transition from the preliminary to the detailed design stage.

ARP is used to analyze the effect of creating a hole for a spar that passes through ribs in the wing. Prior to the use of ARP, the structural optimum function value (f^*) was found to be 20.8309. At the end of 5000 iterations of the ARP algorithm, the probability values corresponding to the projection of the solution with one agent onto the space related to the other 3 agents were all 0.333. As a result, all agents must be included for improvement. The final optimum after using ARP ($f^* = 20.3288$) showed a 2.4% improvement, which is larger than the improvement corresponding cases when individual nodes 1 ($f^* = 20.6882$), 2 ($f^* = 20.7829$), 3 ($f^* = 20.8169$) and 4 ($f^* = 20.7004$) were added. Any other pairwise or triplet combinations of nodes also did not produce any additional improvement. However the final structure obtained was equivalent to projecting to the coalesced node 2-3-4 from 1. When projected onto this node, the final function value improvement is still not superior to the final ARP solution ($f^* = 20.5255$). If the final x^* of the ARP process is not used (directly optimizing a structure with no spar hole, which is similar to our final solution, but by forcing the hole to be filled), the objective function value is further reduced to 20.0991.

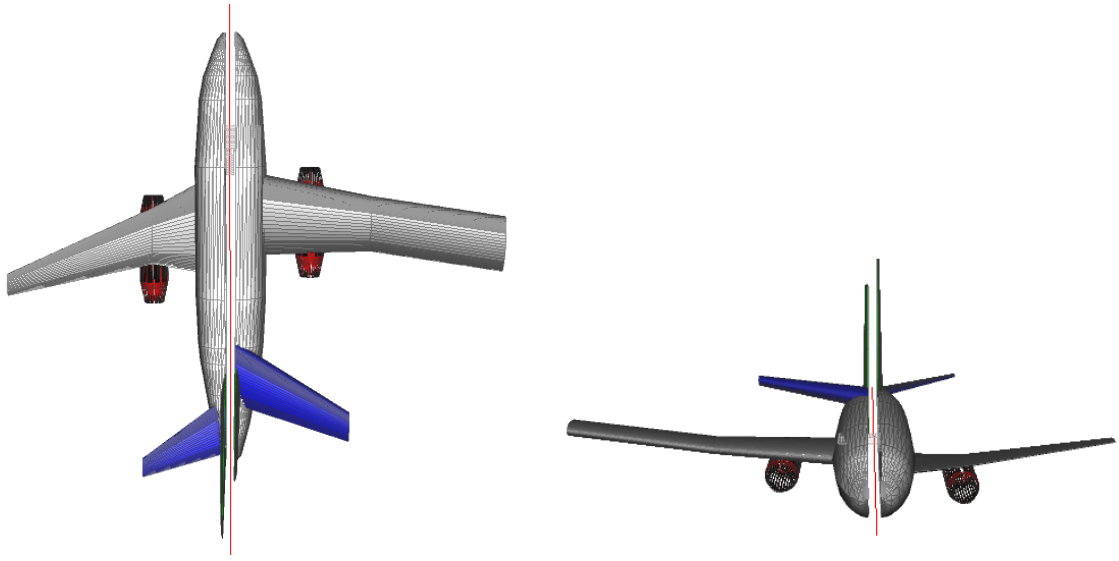
6.9.2 System Optimization

Due to its expensive nature, Airframe noise minimization at the system-level had a maximum function evaluation budget of 5000. To verify the validity of the design, optimization was run 5 times for a given top level requirement. The mean design vector is presented as the final design for comparison with the baseline aircraft (B-737 type airliner). To reiterate, the results shown here only pertain to our given problem formulation which is the design of a noise optimal aircraft in the *approach phase* alone (not a combined result considering other phases of flight such as cruise, take-off etc.). Involving all other phases of flight will improve the overall quality of the aircraft design solution, and will also present a more holistic, and realistic result. This, however is out of the scope of this research.

Since the SoS level requirements are not satisfied by baseline aircraft (through our tests to be discussed next), the aircraft level is triggered to create a new aircraft. This aircraft obtains its requirements from a capability flowdown as described in Chapter 2 . The

objective function value at the system level of the baseline aircraft corresponds to 87.161 dB, which is around the range of values recorded for similar sized aircraft. [166, 168] The improved design is shown in figure 6.10 in comparison to the baseline. A less swept, large aspect ratio wing is selected with the wing, horizontal stabilizer and vertical stabilizer built using NACA 66112, NACA 31312 and NACA 0015 airfoils. At this juncture, we remind the reader that the performance constraints used only pertain to the approach phase. Using a multiple phase of flight type of modeling or optimization may result in a different, more conventional looking aircraft that is optimized for cruise. Once the new airfoil from the subsystem level is generated, it is selected by the optimizer to replace the wing airfoil. The objective function value for the improved aircraft with the wing airfoil being NACA 66112 is 70.421 dB, whereas with the newly generated airfoil is 76.251 dB (these values are used by the top level SoS noise contour module along with the suggested approach velocity). Although the objective function value is larger, the pseudo-objective in the latter case is marginally smaller. This is due to the fact that fewer constraints are active or violated. All configuration constraints were satisfied. Performance constraints that were violated are set by the user, and not to be treated as hard constraints, but rather as guidelines for the optimizer. The constraint to judge the negative slope of the Cm_α curve is found to be active ($+1e-6$ constraint violation), and the downwash constraint is also active ($+1e-12$ constraint violation). The static margin constraint is a soft constraint that may be violated - a calculated static margin of 5% is acceptable since it is close to the value calculated for the baseline (around 5.37%). The tail scrape angle for approach is violated (constraint violation of 0.4941), since fuselage design is not part of our optimization process. This is mainly due to angle of attack for maximum C_L being larger than the difference between the tail-scrape angle and the approach glide slope in the clean configuration.

The increased area, larger aspect ratio wing allows for lower stalling velocity and a higher maximum C_L respectively in the clean configuration. Also, all configuration constraints were satisfied. Refer to the discussion above for active and satisfied performance and other disciplinary constraints in the IDF MDO block. Figure 6.9 shows a side-by-side



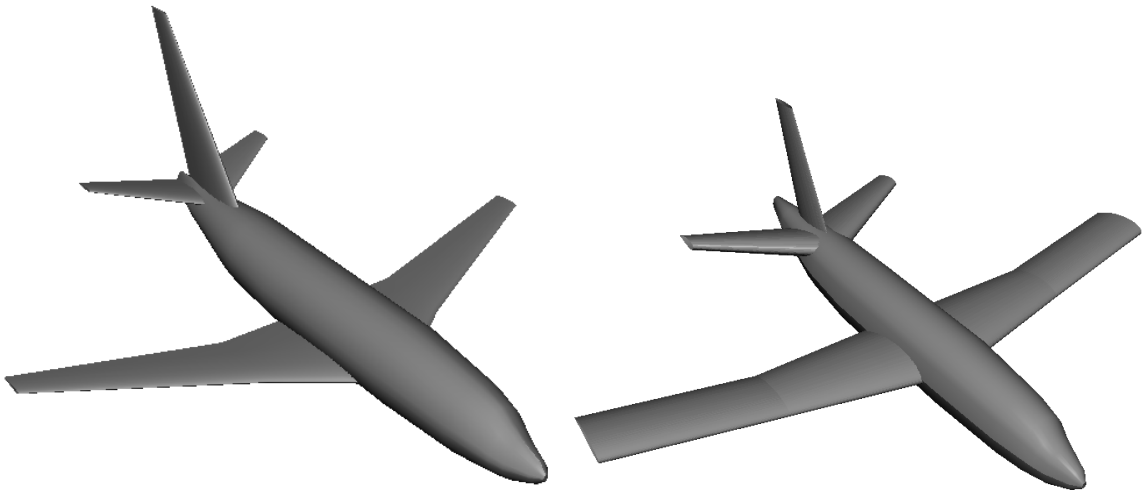
(a) Top view (b) Front view

Figure 6.9.: Comparison of the baseline (left) vs. the improved design (right) for minimized airframe noise - Orthogonal views.

comparison of the baseline (left) and improved(right) designs. A perspective view comparison is also shown below for clarity (see figure 6.10)

6.9.3 SoS Analysis and Optimization

The baseline and improved aircraft designs are flown in trajectories designed in the SoS level of the problem while monitoring noise constraints. Given the lift distributions of the wing, the Augmented Betz block calculates that a single vortex system at 56.37 feet from the symmetry plane of the aircraft will be generated with a strength (Γ_{core}) of $1828.551 ft^2/s$, and a core radius $rc = 4.771 ft$ after 10 seconds of propagation. This is a simplified version of the vortex system generated close to the wing tips, but the reader is reminded that the Augmented Betz method is also capable of producing more complex, multi-vortex systems. Our methodology of generating this system is similar to "effect-based" models used in [186], [187] and [188]. More complex immersed boundary based solvers may be used in the future.



(a) Baseline design (b) Improved design

Figure 6.10.: Comparison of the baseline vs. the improved design for minimized airframe noise - 3D perspective

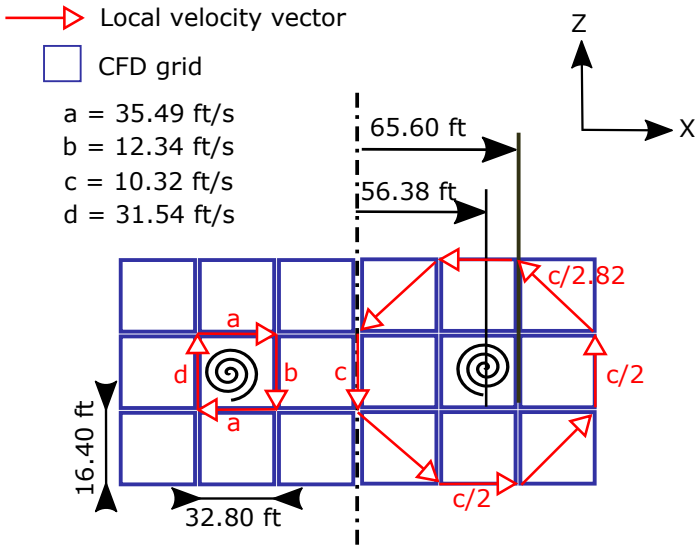


Figure 6.11.: Details of the final vortex distribution on the 'wake snapshot' to be propagated in the SoS CFD box

The CFD solver used on a regular grid with 6363101 cells ($251 \times 251 \times 101$) to resolve a volume of ($2500m \times 2500m \times 500m$). The trajectories and CFD results shown apply to aircraft traveling to the point of intersection before final approach.

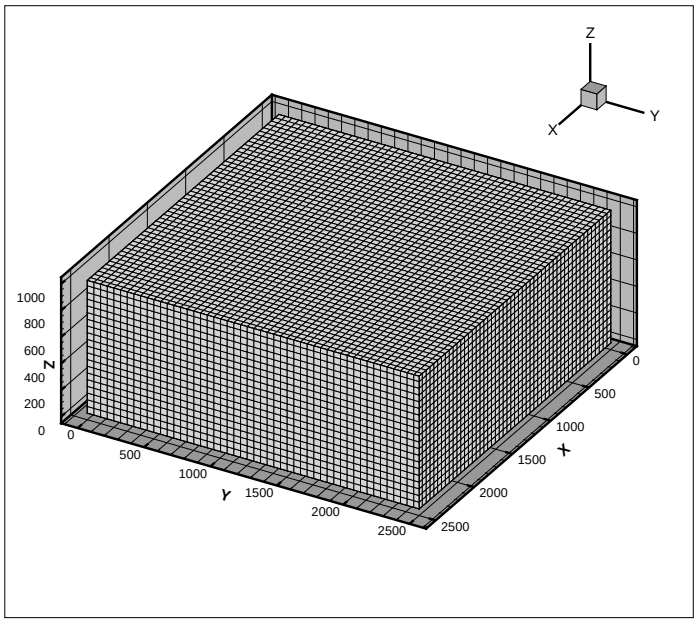


Figure 6.12.: Computational grid used - One in 5 mesh points are plotted as lines on the mesh. X, Y and Z axes are measured in units of meters (m)

A snapshot of a fully developed vortex core is shown (inner and outer) as volume ribbons colored using vorticity magnitude. Observe that a stable, constant magnitude vortex structure is generated. Although pressure fluctuations are minimal (see figure 6.13), a complete picture of the computational domain may be obtained by visualizing slices of vorticity magnitudes along the travel direction (negative y-direction) as shown in figure 6.14.

The path taken by the aircraft approaching runway 17L can also be judged by the vortex distribution contour plot on the X direction contour slice. Since aircraft approaching runway 17R shed vortices that do not interact with aircraft approaching 17L, analyzing the effects of the same would be unnecessary (unless cross-wind effects are present). Given that a follower aircraft "sees" this particular vorticity magnitude distribution ahead of it, the only problem that remains to be solved for a complete solution is the modification of the original time optimal trajectory to the intersection point of the approach course to runway 17R. Velocity gradient Eigenmodes are used to extract vortex cores present in the simulation (refer TECPLOT documentation, www.tecplot.com) This helps us confirm the

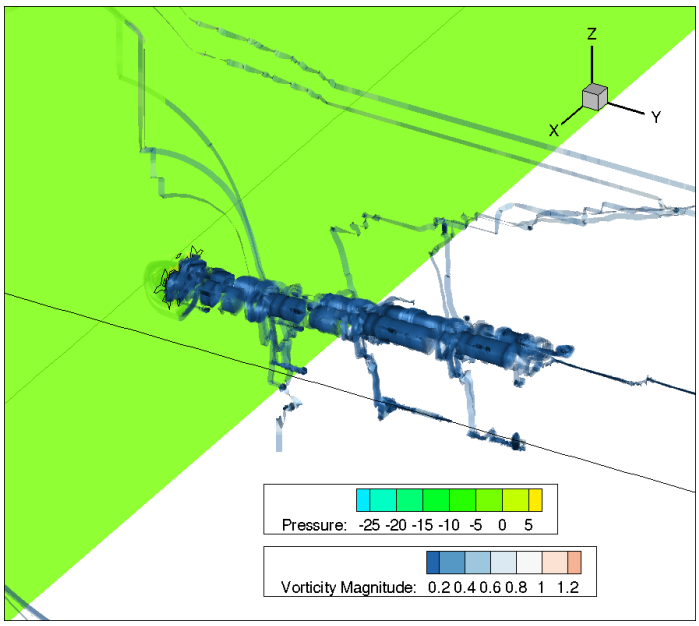


Figure 6.13.: Snapshot of a fully developed vortex core behind the aircraft

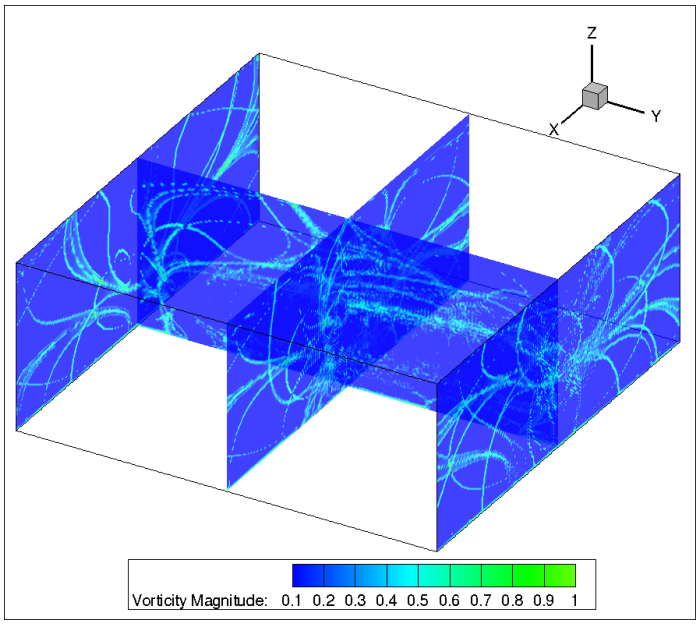


Figure 6.14.: Slices of vorticity magnitude along the travel direction. For the given aircraft, two cores are shed from each side of the wing. In the given figure, the aircraft travels on the central x-slice from the right to left.

presence of stable vortex cores behind the aircraft being simulated. In addition to the vortex core, iso-surfaces of high vorticity are displayed in figure 6.15. This allows us to record areas of high vorticity to be avoided by follower aircraft. As shown in figure 6.15, any follower aircraft must avoid the vortex cores (in red), as well as high vortex regions (identified in blue with iso-surfaces of vorticity magnitude ≥ 0.8 /s, which is less than half the maximum recorded value in the domain 1.723 /s).

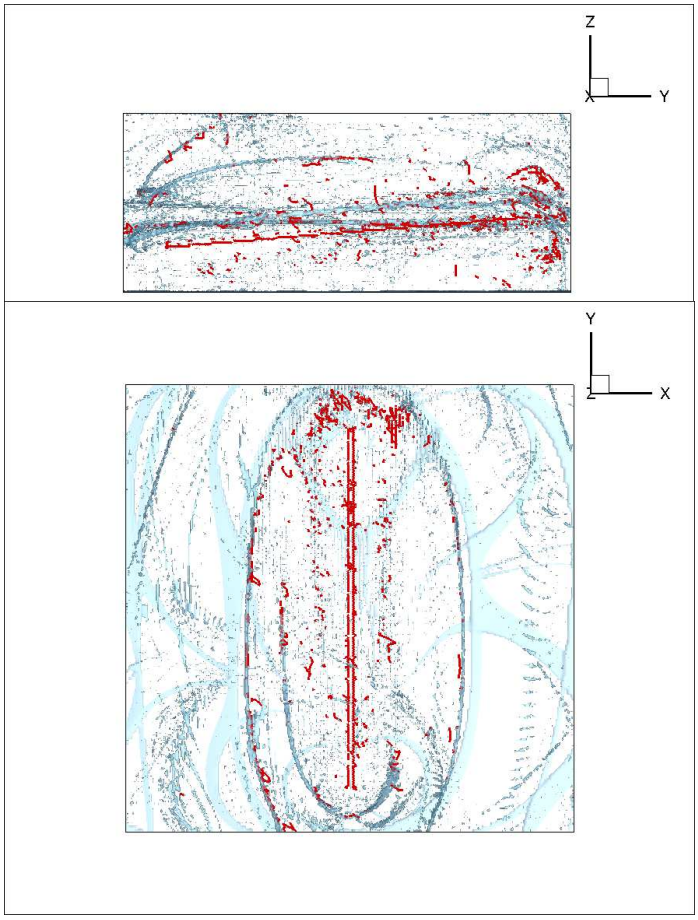


Figure 6.15.: Side view and top view of the computational domain showing extracted vortex cores (in red) and iso-surfaces of vorticity magnitude $= 0.8$

To incorporate this new information from the calculated vortex iso-surfaces, a third objective is added to our multi-objective goal attainment problem. This also demonstrates the flexibility of the hybrid optimal control solution method. Other solvers may also achieve

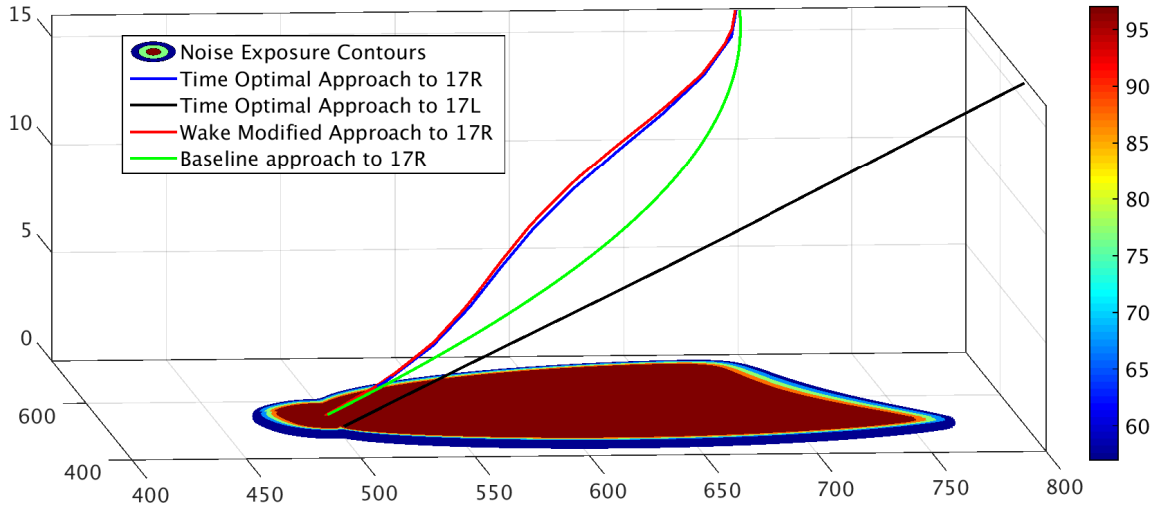
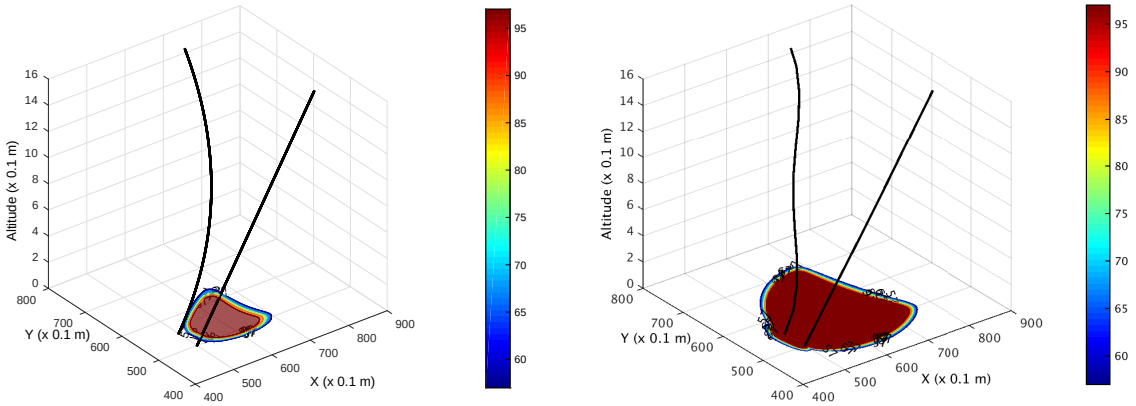


Figure 6.16.: Time optimal and wake modified trajectories to the parallel runways in consideration. Time optimal trajectory to runway 17L is identical to the baseline and is hence not shown here. Noise contour shown corresponds to baseline aircraft flown on new optimal trajectories.

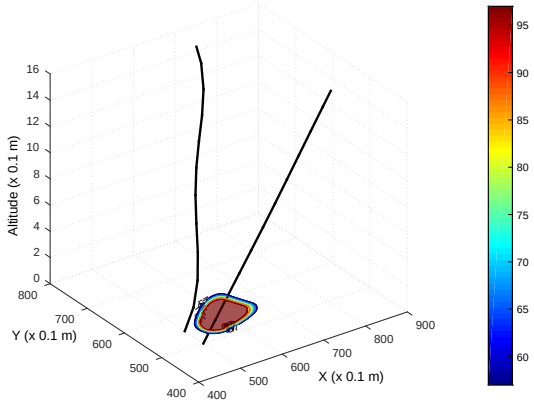
the same effect (say, through continuation), and the ease with which new dimensions of the problem (or new objectives) can be added can vary. The third objective is to maximize the integral of distance to these high cortex centers. Calculation involves measuring the distance from each point on a given trajectory to the closest point on the isosurface. As mentioned earlier, we only expect a minor deviation in course with an associated loss in time (time of flight increased from 29.402 seconds to 29.490 seconds. Figure 6.16 shows all trajectories analyzed in our problem.

Noise intensity comparisons can be made using figure 6.17. The baseline scenario (a) is improved in (b), but is flown using baseline aircraft. As shown in figure 6.17 (b), the size of the noise contours in the pre-approach phase are larger. This provides incentive for the System level (and subsequently the sub-system level) to provide more noise optimal designs. Figure 6.17 (c) shows the wake-modified trajectories flown by the new, improved aircraft.



(a) Baseline trajectories flown by baseline aircraft

(b) Optimal trajectories flown by baseline aircraft



(c) Optimal trajectories flown by optimal aircraft

Figure 6.17.: Trajectory comparison - Optimal aircraft flown on optimal trajectories have the smallest noise contour areas. Coordinates are scaled by a factor of 10 and translated such that the intersection altitude is zero in the new coordinate system

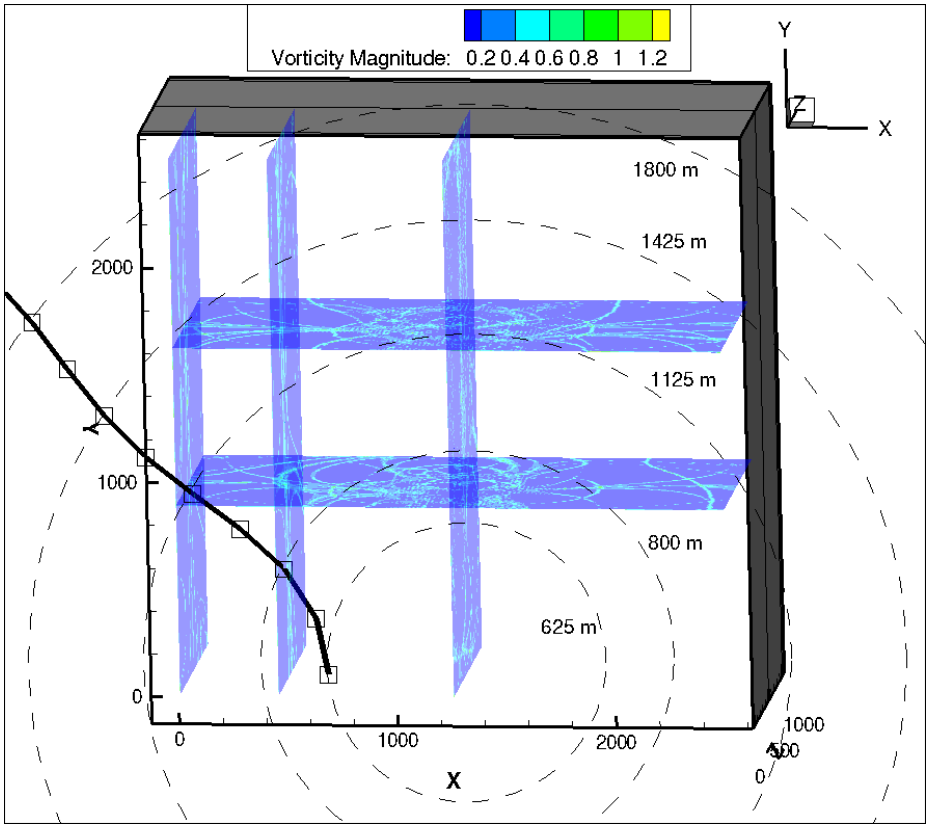


Figure 6.18.: Relative distance of the follower aircraft with respect to the current position of the leader aircraft

To ascertain the follower distance (and indirectly, the capacity of the aircraft queue), spheres of increasing radius and center being the end point of the leader aircraft's trajectory are constructed. Notice in figure 6.18 that the radii are selected based on points on the follower's trajectory to determine the relative position of the follower aircraft. The resolution of these trajectories (points shown as square markers) depends on the number of nodes used in the hybrid optimal control method. At a radius of about $1800m$ ($0.97nm$), we see that the maximum local vorticity magnitude is less than 50% of the overall maximum magnitude. Currently, since there is no association with the follower's position in space and the local vorticity magnitude, an aircraft may be instructed to follow a leader at a farther distance (say, 2.5 nm), but through regions where the local vortex magnitude is close to the maximum recorded magnitude.

Finally, given the change in altitude the constant mach number of the aircraft along the designed trajectory, we analyze if the turbojet engine installed can maintain a constant thrust of 105 KN in the presence of 1) varying atmospheric parameters, and 2) uniform Gaussian fluctuations of total thrust. Recall that the aircraft travels at a constant velocity along the trajectory, the value of which is determined as a parameter as a result of the hybrid optimization problem. As seen in figure A.7 (see Appendix B), although the ram drag increases gradually, mass flow rate is adjusted appropriately by the PID controller (after auto-tuning) to maintain constant thrust. Other outputs of the model such as fuel rate, SFC and net thrust may be used in future studies and analyses.

6.10 Chapter Summary

The proposed SoS optimization framework is demonstrated using a detailed, multi-level, multi-disciplinary aircraft design problem involving the modification, analysis and optimization of aircraft sub-systems, aircraft and approach procedures. More specifically, time optimal trajectories to the intersection points of an approach course to runways 17L and 17R of Austin Bergstrom airport are constructed. Since the baseline aircraft flown on these new trajectories increased the area of noise exposure contours, new aircraft were designed using an evolutionary optimization algorithm (DE-SOM2). Furthermore, sub-system level problems such as the control of the total thrust generated by the turbojet, and the aero-structural optimization of potential new airfoils to be used were solved as part of the process. Thus, we have interpreted and demonstrated the SoS optimization problem by constructing a tree of MDO problems and associated disciplinary analyses that also involve features such as evolving design spaces and interacting optimization problems.

Although the tools and methods used in this chapter yield **products** at each level that are locally optimal or "improved" with respect to a baseline, the **process** of obtaining these solutions through our procedure (including capability flowdown and multi-disciplinary analysis management) is more important. Consider a hypothetical result wherein improving the approach procedure/ trajectory alone (at the SoS level) may have satisfied our require-

ments and noise constraints with the baseline aircraft. In this scenario, there would be no need for improving the aircraft at the System-level, and therefore no need to improve the airfoil used in the sub-system level. Since there is no capability flowdown from the SoS level here, a "solution" to this problem would have only consisted of changes in the SoS level. This differentiates the procedure used here from existing decomposition MDO or decomposition approaches. Additionally, the application problem that we have dealt with here may not be solved *as is* using exiting MDO or decomposition frameworks for two reasons: 1) Our framework operates on a tree of MDO problems, and not a single, level-specific MDO problem, thus allowing inter- and intra-level interactions. This roughly relates to the question - "How does one MDO problem interact with another MDO problem", and 2) Although MDO frameworks may introduce a sense of hierarchy by assuming that the disciplines (in MDO) solve level-specific problems (for example, aircraft level and swarm/fleet level), our inability to use these MDO frameworks as is from literature to solve SoS optimization problems arises from the fact that we are not dealing with the design and optimization of a single system, but the assembly and operation of a group of systems, that may each be designed by an MDO method. In the example problem shown in this chapter, IDF and ECO frameworks were used to design these sub-systems or systems (frequently called products or objects). These systems contribute to the assembly of a system at a higher level in the SoS hierarchy.

Our discussion above highlights important points about the problem structure itself. However, the solution methodology used in Application Problem 2 (this chapter) is different from that used in Application problem 1. In Application Problem 1 (chapter 3), we allowed the progression of each of the three levels in the hierarchy in parallel through the use of SPMD. This would mean that level constantly checks for updates in the corresponding libraries of systems in lower levels. These updates happen independently, as though sub-contractors would design new systems, and add them to their brochure of available systems for contractors or higher-level manufacturers to pick from. In Application problem 2, however, the aircraft level is triggered into action through a capability flowdown if and only if an improved approach procedure with baseline aircraft does not satisfy some

requirements or constraints. In this scenario, a sub-contractor is incentivized to produce a new product only if none of the other products in his brochure of available systems or products satisfies the needs (or requirements) of a higher level contractor. This brings the solution methodology closer to practical information flow in realistic design projects in the industry.

7. Conclusions and Future work

A procedure to integrate and solve a tree of level-specific MDAO problems through a generalized mathematical template is presented. The framework incorporates special features commonly encountered in the SoS optimization context. Managing inter- and intra-level interactions in a hierarchy of MDO problems is achieved using features from Platform-based design and an adaption of the hierarchical complexity metric. Other features such as the use of expensive black-box functions in optimization, optimization in evolving design spaces, and the use of multi-fidelity tools have spurred the need for specialized algorithms and methods to be created (DE-SOM, ARP and a modified version of VoI). Research conclusions and future work are discussed in the following sections:

7.1 SoS Optimization Framework

The work presents a design paradigm that serves as a generic template for SoS design optimization. This portion of the work involved the following topics:

- Used a **Platform-based SoS** design and optimization framework for
  - Assembling optimal sub-systems, novel aircraft systems and SoS
  - **Hierarchical complexity** guided design (high performance while product & process is *controlled*)
- Demonstrated solution for a **unique problem formulation**:
  - Multi-level, multidisciplinary, multi-fidelity
  - Evolving design spaces at multiple levels (progressing in parallel)
  - Solution strategy for cases where levels do not have a primary objective

- **Complexity Management**: Complexity guided search was established in the case of SoS optimization problems with both a time, as well as computational resource budget. Here complexity involves both, operational complexity of the components at each level, and computational complexity associated with resources. We are required to test the proposed Hierarchical complexity metric for managing a sample process involving design and optimization of SoS. The metric, along with Value of Information (VoI) used for multi-fidelity analysis, can be improved or reformulated as required.

- **SoS System Simulator** - Developed an SoS System Simulator (physics based ABM) to test the SoS in the face of operational constraints. The aircraft designed in the System level are 'flown' in the SoS level to judge mission feasibility.

- **Challenge Problems** - Two unique application problems were solved to demonstrate the various facets of this research. Although the two problems are instances of the same SoS optimization framework, they are different in terms of scale, the kind of algorithms use and the nature of the evolving design space.

- **Hardware and software tools** - Set up computational tools and frameworks that are used to solve two completely different application problems. In the first application problem, Single Program Multiple Data (SPMD) was used to demonstrate an embarrassingly parallel solution method when there are limited number of processors and a constrained time budget. On the other hand, multiple workers were sequentially used to solve the second application problem. Choice of hardware tools and parallel processing architectures suit the time scale of problem itself (synchronous levels versus asynchronous levels).

7.2 Algorithm Development

Apart from presenting a standardized mathematical framework for the solution of SoS optimization problem, this work also improves the state of the art for efficient analysis of

a multi-level, evolving design space that is characteristic to SoS problems. Although we could take advantage of the several optimization algorithms existing in literature, the challenge problem described here is special, and can not be solved efficiently by any existing framework. Algorithmic advancements that address the special features germane to SoS optimization problems are summarized below:

- We extend our existing DE-SOM algorithm to handle the following special cases:
  - Discrete-continuous combination of variables forming the design space,
  - Stochastic program (objective function or constraints are not deterministic),
  - multi-level, evolving design space.

  As a result, we have rigorously tested and widely applicable evolutionary optimization algorithms for (**DE-SOM** and **DE-SOM2**) **expensive objective functions**
- Demonstrated the use of a provably convergent algorithm, Adaptive Random Projections (ARP) for use in Evolving design spaces of type 2, Type 1 being Numerical Continuation, which is well known in Topology optimization literature.
- Introduced a Hybrid Optimal Control method that converts a trajectory optimization problem to a form that can be utilized by existing multi-objective, non-linear optimizers. Comparisons with trajectories and control histories generated by other direct and indirect optimal control solvers will be presented in a future paper. At the time of writing this thesis, this algorithm is still a work in progress and has already been subject to various improvements to yield more realistic, physical trajectories.

7.3 Submitted Journal Papers

The following papers in relation to this research work have been submitted to / are in review with journals mentioned below:

1. Self-Organizing Maps based Differential Evolution for Resource Intensive Optimization, *Journal of Global Optimization*, May 2015
2. Dual Averaging with Adaptive Random Projection (ARP) for Solving Evolving Distributed Optimization Problems , *Journal of Optimization Theory and Applications*, June 2015
3. Application of Multidisciplinary System-of-systems Optimization to an Aircraft Design Problem , *Wiley Systems Engineering Journal*, July 2015
4. Dual Phase Consensus Algorithm for Distributed Sensor Management, *IEEE Transactions on Aerospace and Electronic Systems*, July 2015

APPENDICES

A. APPENDICES

A.1 Appendices for Chapter 1

Please see next page for large infographic.

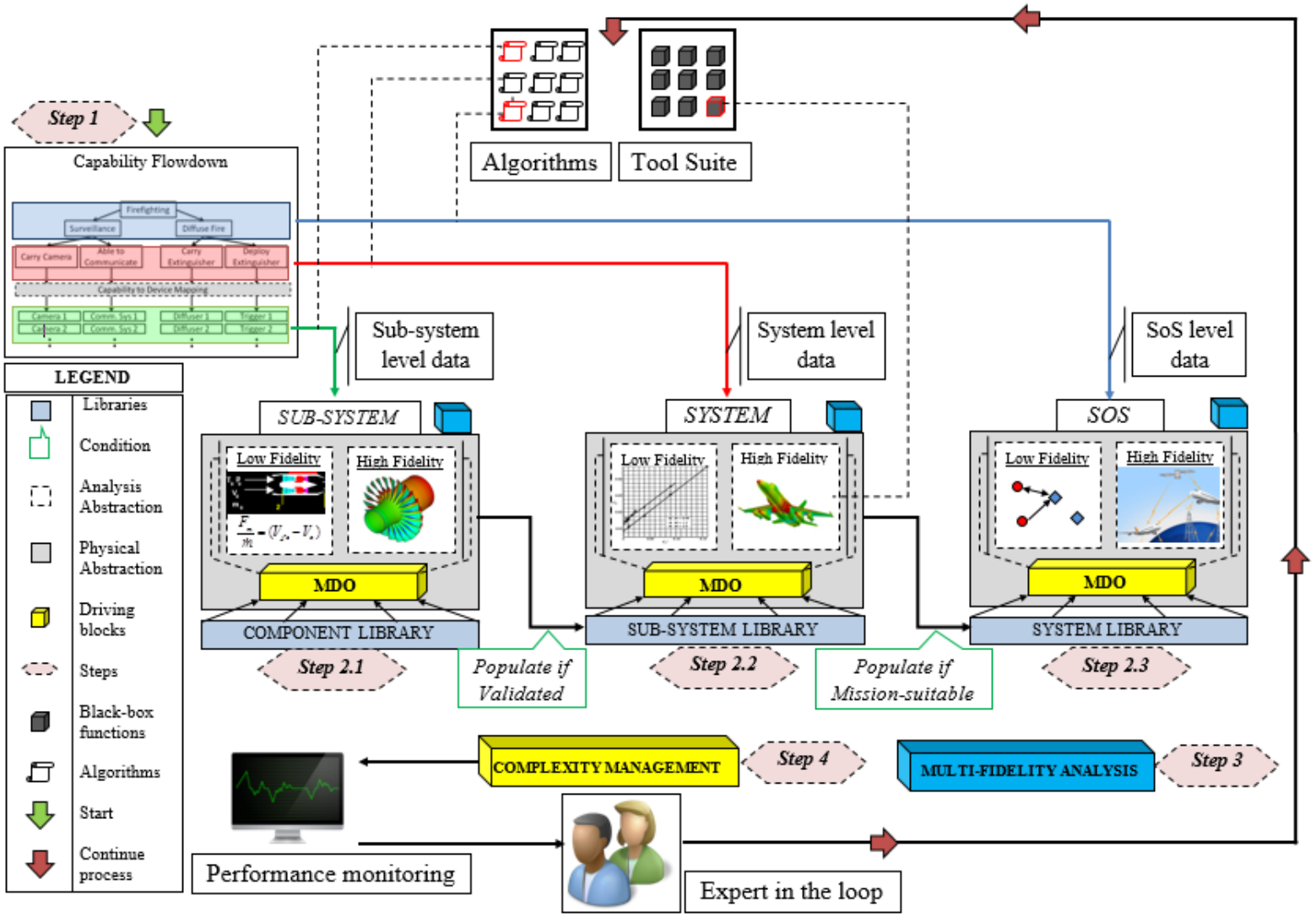


Figure A.1.: **Appendix A** for Chapter 1 : Info-graphic describing the framework envisioned

A.2 Appendices for Chapter 3

Appendix A - Algorithms

Algorithm DE

Require: Initial bounds contain targeted optimum

Initialization(); Generate uniformly distributed random population
while termination condition not met **do**
  for $i = 1$ **to** NP **do**
    Select random indexes $r1$,$r2$ and $r3$ to be different from each other and from the index i
    $v_i^G = x_{r1}^G + F \times (x_{r2}^G - x_{r3}^G)$
    $j_{rand} = rand(1, N)$
    for $j = 1$ **to** N **do**
      if $(rand(0,1) \leq CR$ **or** $j == j_{rand})$ **then**
        $u_{i,j}^G \leftarrow v_{i,j}^G$
      else
        $u_{i,j}^G \leftarrow x_{i,j}^G$
      end if
    end for
    if $(f(u_{i,j}) \leq f(x_{i,j})$ **then**
      $x_{i,j}^{G+1} \leftarrow u_{i,j}^G$
    end if
  end for
end while

Algorithm DE-SOM

Require: Initial bounds contain targeted optimum

Initialization(); Generate uniformly distributed random population
Set p,CR, F
while termination condition not met **do**
  if $rand \leq p$ **then**
    Use DE
  else
    Use DE-SOM
  end if
  for $i = 1$ **to** NP **do**
    Select random indexes $r1$,$r2$ and $r3$ to be different from each other and from the index i
    $v_i^G = x_{r1}^G + F \times (x_{r2}^G - x_{r3}^G)$
    $j_{rand} = rand(1, N)$
    for $j = 1$ **to** N **do**
      if $(rand(0,1) \leq CR$ **or** $j == j_{rand})$ **then**
        $u_{i,j}^G \leftarrow v_{i,j}^G$
      else
        $u_{i,j}^G \leftarrow x_{i,j}^G$
      end if
    end for
    if $x_{i,j}^{G+1}$ is in convex hull of all $x_{i,j}^G$ **and** Use DE-SOM **then**
      $x_{i,j}^{G+1} \leftarrow u_{i,j}^G$
    else if Use DE **and** $(f(u_{i,j}) \leq f(x_{i,j})$ **then**
      $x_{i,j}^{G+1} \leftarrow u_{i,j}^G$
    else
      $x_{i,j}$ remains the same
    end if
  end for
  {Replace under-performers with elite members from SOM}:
  Perform SOM to obtain weights
  Rank-order Neurons
  Top $e\%$ neurons become members
end while

Algorithm DE-SOM2

Require: Initial bounds contain targeted optimum

Initialization(); Generate uniformly distributed random population
Set p_0,γ_{p1},γ_{p2},F_0,F_{max},F_{min}, CR
while termination condition not met **do**
  if $rand \leq p_k$ **and** $var(\text{population}) \leq 1e-1$ **then**
    Use DE
  else
    Use DE-SOM
  end if
  for $i = 1$ **to** NP **do**
    Select random indexes $r1$,$r2$ and $r3$ to be different from each other and from the index i
    $v_i^G = x_{r1}^G + F \times (x_{r2}^G - x_{r3}^G)$
    $j_{rand} = rand(1,N)$
    for $j = 1$ **to** N **do**
      if $(rand(0,1) \leq CR$ **or** $j == j_{rand})$ **then**
        $u_{i,j}^G \leftarrow v_{i,j}^G$
      else
        $u_{i,j}^G \leftarrow x_{i,j}^G$
      end if
    end for
    if $x_{i,j}^{G+1}$ is in convex hull of all $x_{i,j}^G$ **and** Use DE-SOM **then**
      $x_{i,j}^{G+1} \leftarrow u_{i,j}^G$
    else if $x_{i,j}^{G+1}$ is MVCE of $x_{i,j}^G$ **and** Use DE-SOM **then**
      $x_{i,j}^{G+1} \leftarrow u_{i,j}^G$
    else if Use DE **and** $(\hat{f}(u_{i,j}^G) \leq f(x_{i,j}^{G-1})$ **then**
      $x_{i,j}^{G+1} \leftarrow u_{i,j}^G$
    else if Use DE **and** $(f(u_{i,j}) \leq f(x_{i,j})$ **then**
      $x_{i,j}^{G+1} \leftarrow u_{i,j}^G$
    else
      $x_{i,j}$ remains the same
    end if
  end for
end while

Algorithm DE-SOM2 (continued...)

Require: continue from line 32 in while loop

{Replace under-performers with elite members from SOM}:

Perform SOM to obtain weights

Rank-order Neurons

Top $e\%$ neurons become members

if mod(k,l)=0 **then**

$$p_{k+l} = \begin{cases} min(p_k \times \gamma_{p1}, 1) & \text{if } favg_{k+l} \geq favg_k \\ max(p_k / \gamma_{p2}, 0) & \text{if } favg_{k+l} < favg_k \end{cases}$$

end if

if mod(k,m)=0 **then**

$$F_{k+m} = \begin{cases} min(F_k / rand, F_{max}) & \text{if } favg_{k+l} \geq favg_k \\ max(F_k \times rand, F_{min}) & \text{if } favg_{k+l} < favg_k \end{cases}$$

end if

Appendix B - Convergence of DE-SOM

The following supplementary material is assembled for the interested reader to obtain detailed information about the mathematical derivations and formulae presented in the main text. We also provide additional results from the application problems presented therein.

Nomenclature

α = angle of attack

ε = displacement

σ = stress

Cd = section drag coefficient

Cl = section lift coefficient

CR = crossover probability

F = mutation factor

G = generation number

f = objective function

kn = dimension of self organizing map

L = polynomial of a random variable

M = mach number

N = dimension of the design variable

NF = number of function evaluations

NP = size of population

p = switching probability

q = probability of containment

$rand$ = random number $\in [0,1]$

Re = Reynolds number

u = trial vector

v = mutant vector

w = neuron vector

Detailed Derivations

Expected Value of the Trial Vector

Let x_i be the i'th population member in the final stages of the DE-SOM algorithm. We assume that all the members of the population are concentrated around the optimum. For the *DE/rand/1* version of the algorithm, the mutant vector v_i is generated as follows:

$$v_i = x_{r1,i} + F \times (x_{r2,i} - x_{r3,i}) \tag{A.1}$$

where F is the mutation scaling factor and the x_{rk} terms with $k \in \{1,2,3\}$ are mutually exclusive random vectors drawn without replacement from the population of the current generation. Since the vectors x_{rk} are independent of each other, $P(x_{ri} \mid x_{rj}) = P(x_{ri})$. The trial vector u_i is a result of the crossover operation given by

$$u_i = \begin{cases} v_i & \text{if } rand \leq CR \vee j = j_{rand} \\ x_i & \text{otherwise} \end{cases} \tag{A.2}$$

Thus, the probability of the trial vector u_i inheriting components of the mutant vector v_i is given by CR. Thus the expected value of the trial vector is given by

$$E(u_i) = (1 - CR) \cdot (x_i) + (CR) \sum_{i=1}^{NP} \sum_{i=1}^{NP} \sum_{i=1}^{NP} \{P(x_{r1}) \cdot P(x_{r2}) \cdot P(x_{r3}) \times (v_i)\} \tag{A.3}$$

where $P(x_{rk})$ represents probability of picking a random vector from the population to form the mutant. Since three vectors $r1$, $r2$ and $r3$ are independently selected, $P(x_{rk}) = \frac{1}{NP}$. Also, the mutant vector is given by
$v_i = x_{r1} + F \cdot (x_{r2} - x_{r3})$.

$$\begin{aligned}\therefore E(u_i) &= (1-CR)\cdot x_i + \frac{CR}{NP^3}\cdot\sum_{i=1}^{NP}\sum_{i=1}^{NP}\sum_{i=1}^{NP}\{x_{r1}+F(x_{r2}-x_{r3})\} \\ &= (1-CR)\cdot x_i + \frac{CR}{NP^3}\cdot\sum_{i=1}^{NP}\sum_{i=1}^{NP}\sum_{i=1}^{NP}x_1 + F(\sum_{i=1}^{NP}\sum_{i=1}^{NP}\sum_{i=1}^{NP}x_2 - \sum_{i=1}^{NP}\sum_{i=1}^{NP}\sum_{i=1}^{NP}x_3) \\ &= (1-CR)\cdot(x_i) + \frac{CR}{NP}\sum_{i=1}^{NP}(x_i)\end{aligned}$$

Thus, with mean vector $x_{av} = \frac{1}{NP}\sum_{i=1}^{NP}(x_r k)$, we get:

$$E(u_i) = (1-CR)\cdot(x_i) + CR\cdot(x_{av}) \tag{A.4}$$

Desired velocity of population members

As in Dasgupta *et al.*, let us assume that DE is a process that occurs in continuous time, and the decision of selecting x_i or u_i is made using the heavyside unit step function, given by [189]

$$w(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Now, change in the position vector of a DE population member is given by $\Delta x_i = u_i - x_i$. Based on whether u_i or x_i is preferred, we need to use the heavyside step function to switch to the desired output. We repeat part of the proof in [189] and extend it to our algorithm. Consider the expression:

$$\frac{\Delta x_i}{\Delta t} = w\{\frac{f(x_i) - f(x_i + \Delta x_i)}{\Delta t}\}\cdot(u_i - x_i) \tag{A.5}$$

Equation A.5 has the following implications : If $f(x_i) \leq f(x_i - \Delta x_i)$, $\Delta x_i = 0$ and thus there is no change in the vector x_i. However if $f(x_i) \geq f(x_i - \Delta x_i)$, $\Delta x_i = u_i - x_i$, and so the vector x_i becomes u_i in the next generation. As we have established in the discussion

above, DE-SOM achieves the same effect as a function value comparison without actual function evaluations. For a continuous time approximation, the time step $\Delta t \to 0$. Thus, from Equation A.5 we get:

$$\begin{aligned} \lim_{\Delta t \to 0} \frac{\Delta x_i}{\Delta t} &= \lim_{\Delta t \to 0} w\left(\frac{f(x_i) - f(x_i + \Delta x_i)}{\Delta t} \cdot \frac{\Delta x_i}{\Delta t}\right) \cdot (u_i - x_i) \\ \implies \frac{dx_i}{dt} &= w\left(-f'(x_i)\frac{x_i}{dt}\right)(u_i - x_i) \end{aligned} \tag{A.6}$$

For some large value of k, we can write an approximation for the heavyside step function as $w(a) \approxeq \frac{1}{2} + \frac{k}{4}a$. [189]

$$\begin{aligned} \therefore \frac{dx_i}{dt} &= \left(\frac{1}{2} - \frac{k}{4}f'(x_i)\frac{x_i}{dt}\right)(u_i - x_i) \\ \implies \frac{dx_i}{dt} &= \frac{1}{2}(u_i - x_i)\left(1 + \frac{k}{4}f'(x_i)(u_i - x_i)\right)^{-1} \\ &\approxeq -\frac{k}{8}(u_i - x_i)^2 f'(x_i) + \frac{(u_i - x_i)}{2} \end{aligned} \tag{A.7}$$

But $f'(x_i) = 0$ at optimum. Using this information, and taking expected value of both sides of equation A.6, we get the expected value of the velocity of a vector x_i -

$$\begin{aligned} E(\frac{dx_i}{dt}) &= \frac{1}{2}E(u_i - x_i) \\ &= \frac{1}{2}(E(u_i) - E(x_i)) \\ &= \frac{1}{2}((1 - CR) \cdot (x_i) + CR \cdot (x_{av}) - x_i) \\ &= \frac{CR}{2}(x_{av} - x_i) \end{aligned} \tag{A.8}$$

Expected Variation in Population

Recall that the mutant vector is v_i and the trial vector is u_i. We know that $E(Var(v)) = E(\overline{v^2}) - E(\overline{v}^2)$. Since the mutation step is the same as a regular DE, we use the expression for $E(Var(v))$ from Zaharie *et al.* [190], which is given by equation A.9

$$E(Var(v)) = \left(2F^2 + \frac{NP-1}{NP}\right) \tag{A.9}$$

In DE-SOM, the selection step is given by equation A.10

$$u_i = \begin{cases} v_i & \text{if } v_i \in conv(x_{1\to NP}) \\ x_i & \text{otherwise} \end{cases} \tag{A.10}$$

Let the probablity of a point's presence in a convex hull be given by q (referred to as Probability of Containment). The expected value of the variance of the trial vector may be given by $E(Var(u)) = E(\overline{u^2}) - E(\overline{u}^2)$. Now,

$$\begin{aligned} E(\overline{u^2}) &= \frac{1}{NP}\sum_{i=1}^{NP} E(u_i^2) \\ &= \frac{1}{NP}\sum_{i=1}^{NP} \left((1-q)\cdot E(x_i^2) + q\cdot E(v_i^2)\right) \\ &= (1-q)\overline{x^2} + (q)E(\overline{v^2}) \end{aligned} \tag{A.11}$$

$$\begin{aligned} E(\overline{u}^2) &= \frac{1}{NP^2} E\left((\sum_{i=1}^{NP} u_i)^2\right) \\ &= \frac{1}{NP^2}\left(E(\sum_{i=1}^{NP} u_i^2) + E(\sum_{i\neq j} u_i \cdot u_j)\right) \\ &= \frac{1}{NP}E(\overline{u_i^2}) + \frac{1}{NP^2}E(u_i)\cdot E(u_j) \end{aligned} \tag{A.12}$$

But we know that, $E(u_i)E(u_j) = [(1-q)E(x_i) + qE(v_i)] \times [(1-q)E(x_j) + qE(v_j)]$, and $E(v_i) = E(x_{r1} + F(x_{r2} - x_{r3})) = \overline{x}$. Thus we can calculate the value of $E(u_i)E(u_j)$ in equation A.12:

$$\begin{aligned}\sum_{i\neq j} E(u_i)E(u_j) &= \sum_{i=1}^{NP}(E(u_i))^2 - \sum_{i=1}^{NP} E(u_i^2) \\ &= [\sum_{i=1}^{NP}(1-q)x_i + q\overline{x}]^2 - \sum_{i=1}^{NP}[(1-q)x_i + q\overline{x}]^2 \\ &= (NP^2 - 2(NP) + (NP)q^2)\overline{x}^2 - NP(1-q)^2\overline{x^2}\end{aligned}$$

Substituting this value in equation A.12 we get

$$E(\overline{u}^2) = \frac{1}{NP}E(\overline{u^2}) + (NP^2 - 2(NP) + (NP)q^2)\overline{x}^2 - NP(1-q)^2\overline{x^2} \tag{A.13}$$

From equation A.11, A.12 and A.13, we can assemble $E(Var(u)) = E(\overline{u^2}) - E(\overline{u}^2)$ to get:

$$\begin{aligned}E(Var(u)) &= (1-\frac{1}{NP})E(\overline{u^2}) - \left(1 - \frac{2q}{NP} + \frac{q^2}{NP}\right)\overline{x}^2 + (\frac{(1-q)^2}{NP})\overline{x^2} \\ &= \left(\frac{NP-1}{NP}\right)\left((1-q)\overline{x^2} + p\left(2F^2\frac{NP}{NP-1} + 1\right)\overline{x^2} - \frac{2F^2(NP)}{NP-1}\right) \\ &- \left(1 - \frac{2q}{NP} + \frac{q^2}{NP}\right)\overline{x}^2 + \left(\frac{(1-q)^2}{NP}\right)\overline{x^2}\end{aligned}$$

This can be simplified to obtain the following result:

$$\begin{aligned}E(Var(u)) &= \left(\frac{q^2}{NP} + \left(2F^2 - \frac{2}{NP}\right)q + 1\right)Var(x) \\ &= L_q \cdot Var(x)\end{aligned} \tag{A.14}$$

Where equation A.14 relates the expected value of the variance of the final trial vector to the variance of the initial population through L_q, a polynomial in q.

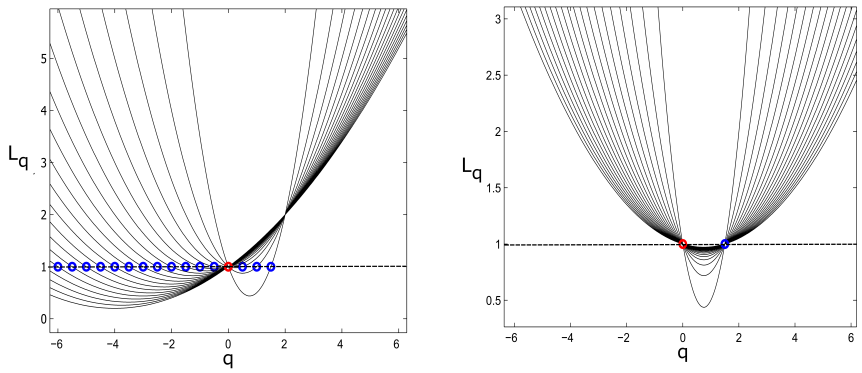
Parameter bounds

Suppose the convex n-dimensional polytope decreases in hyper volume by a quantity ΔV with respect to its original volume V. The quantity q is simply the probability of a point's presence in the changed hyper-volume, which is given by $q = \frac{\Delta V}{V}$. In figure 2, replacing the DE member at point '2' with an internal point causes the volume to decrease (ΔV). A small change in hyper-volume implies that the newly generated point can most probably be found in the new hyper-volume. Thus, $q = 0$ corresponds to a condition where there is a variance in the internal population, but the convex hull (or its volume) remains the same. Minimum change in variance may be obtained by minimizing the polynomial L_q, which is given in equation A.15.

$$\begin{aligned} Min(L_q) &\implies \frac{d}{dq}L_q = 0 \\ &\implies \frac{d}{dq}\left(\frac{q^2}{NP} + \left(2F^2 - \frac{2}{NP}\right)q + 1\right) = 0 \\ &\implies \left(\frac{2q}{NP} + 2F^2 - \frac{2}{NP}\right) = 0 \\ &\implies q* = 1 - F^2(NP) \end{aligned} \tag{A.15}$$

At the minimum point of the polynomial L_q, the condition $q > 0$ must be satisfied for the desired effect of continuously decreasing volume of the convex hull. Thus we can conclude that $(1 - (F^2)NP) > 0$ or the range of the mutation factor F must be given as $F \in (0, \frac{1}{\sqrt{NP}}])$.

There is a need to adapt the value of F according to the dimension of the members in the population. Figure A.2a shows the effect of holding the value of F fixed (*Case A*). For a small number of F values corresponding to certain integer values of NP, there is a valid region of q from $q = 0$ to $q = 2 - 2F^2 \cdot NP$ that will generate trial vectors that will land in the convex hull. The points corresponding to negative values of q are invalid. Note that any q value that is on a curve under the $Lq = 1$ line is valid. In the example shown, NP is varied from 1 to 20 with a fixed value of $F = 0.5$. Suppose we adapt the parameter F according to the size of the population as $F = h \cdot (1/\sqrt{NP})$, where $h \in (0, 1]$. At $h = 0$,



(a) Case A (b) Case B

Figure A.2.: *Case A:* Plot of L_q with varying NP and $F = 0.5$. *Case B:* Plot of L_q with varying NP and $F = h \cdot \frac{1}{\sqrt{NP}}$. Here $h = 0.5$

$F = 0$ and hence the gap between the two roots $q = 0$ and $q = 2 - 2F^2(NP)$ is maximum. This is not practical as there is no mutation. A value of $h = 1$ implies that the roots coincide at $q = 0$. We need to choose a value of h such that these extreme cases are not encountered. Shown in figure A.2b *Case B* is where $h = 0.5$. While solving a particular problem (fixed population), F may be varied within the valid region under $L_q = 1$. Comparing with figure A.2a, we can see that adapting the value of F as $F = h \cdot (1/\sqrt{NP})$ forces all the curves to have valid regions of q values with a corresponding L_q under 1.

Appendix C - Benchmark Sets

Benchmark Set 1

Table A.1: List of functions used in the benchmark set 1 with the corresponding global optima. Also, the bounds (or extant of the search space) in each direction is shown)

No.	Name	Function	f*	Bounds
f1	Ackley's Function	$-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}cos(2\pi x_i))+20$	0	$[-2,2]$
f2	Sphere Function	$\sum_{i=1}^{n}x_i^2$	0	$[-2,2]$
f3	Rosenbrock's Function	$\sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2+(x_i-1)^2\right]$	0	$[-2,2]$
		$\left(1+(x_1+x_2+1)^2\left(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right)$		
f4	Goldstein-Price Function	$\left(30+(2x_1-3x_2)^2\left(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right)\right)$	3	$[-3,3]$
f5	Levi Function	$\sin^2(\pi x_1)+\sum_{i=1}^{n-1}(x_i-1)^2(1+10\sin^2(\pi x_{i+1}))+(x_n-1)^2$	0	$[-5,5]$
f6	Three-hump Camel Function	$2x_1^2-1.05x_1^4+\frac{x_1^6}{6}+x_1x_2+x_2^2$	0	$[-5,5]$
f7	Easom Function	$-\cos(x_1)\cos(x_2)\exp\left(-\left((x_1-\pi)^2+(x_2-\pi)^2\right)\right)$	-1	$[-10,10]$
f8	Beale's Function	$(1.5-x_1+x_1x_2)^2+\left(2.25-x_1+x_1x_2^2\right)^2+\left(2.625-x_1+x_1x_2^3\right)^2$	0	$[-4.5,4.5]$
f9	Booth's Function	$(x_1+2x_2-7)^2+(2x_1+x_2-5)^2$	0	$[-10,10]$
f10	Matya's Function	$0.26\left(\sum_{i=1}^{n}x_i^2\right)-0.48\prod_{i=1}^{n}(x_i)$	0	$[-10,10]$
f11	Griewank's Function	$1+\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}cos\left(\frac{x_i}{\sqrt{i}}\right)$	0	$[-5,5]$
f12	Rastrigin's Function	$An+\sum_{i=1}^{n}\left(x_i^2-A\cos(2\pi x_i)\right)$	0	$[-5.12,5.12]$
f13	Schwefel's Function	$\sum_{i=1}^{n}-x_i\cdot\sin(\sqrt{\lvert x_i\rvert})$	-837.9658	$[-500,500]$
f14	Moved-axis parallel ellipsoid	$\sum_{i=1}^{n}5i\cdot x_i^2$	0	$[-4,4]$
f15	Michalewicz's Function	$-\sum_{i=1}^{n}\sin(x_i)\cdot\left(\sin\left(\frac{i\cdot x_i}{\pi}\right)\right)^{20}$	-1.8983	$[-15,15]$

Benchmark Set 2

Appendix D - Optimal Airfoils of GA and DE

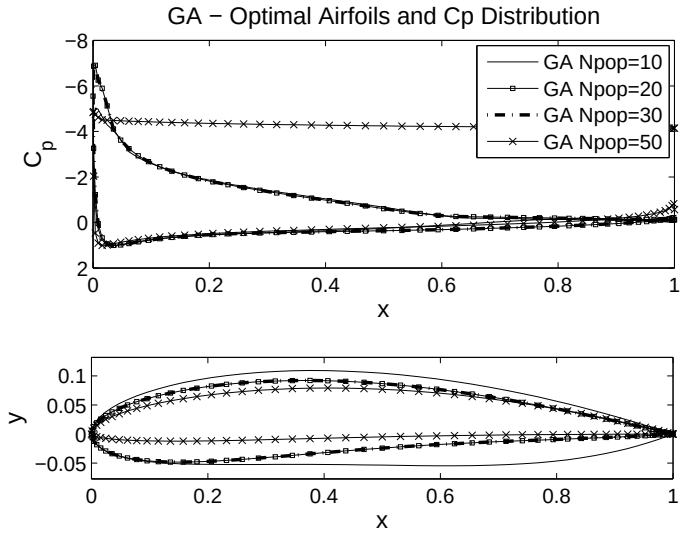


Figure A.3.: Airfoils belonging to the different population cases that are results of the GA algorithm distinguished by their Cp distribution.

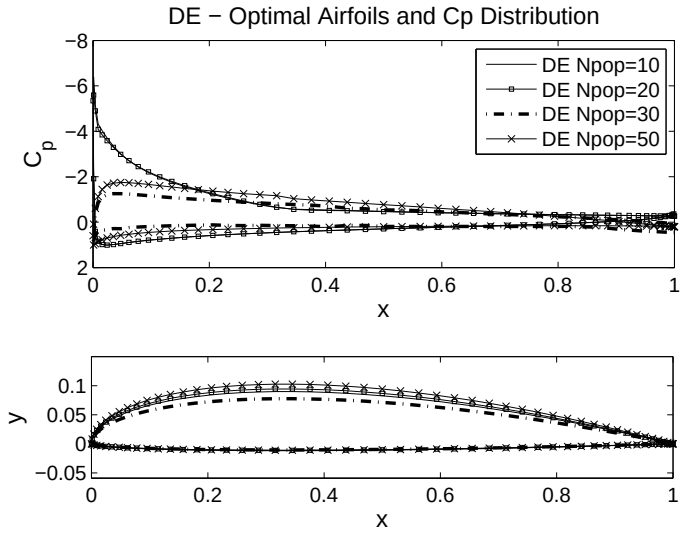


Figure A.4.: Airfoils belonging to the different population cases that are results of the DE algorithm distinguished by their Cp distribution.

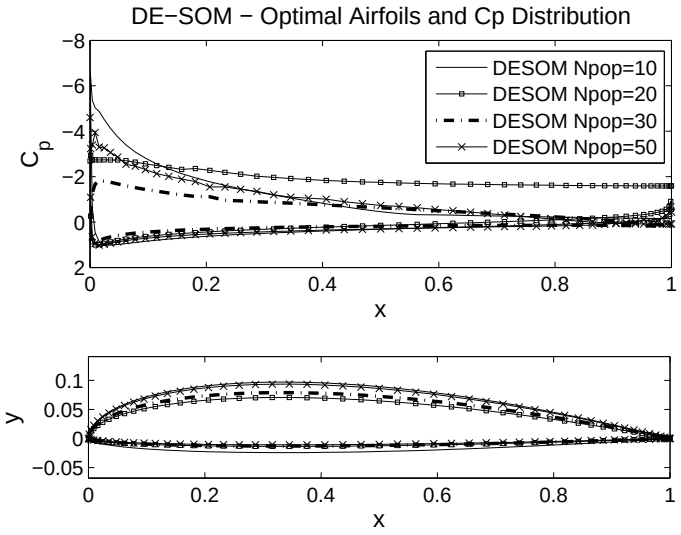


Figure A.5.: Airfoils belonging to the different population cases that are results of the DE-SOM algorithm distinguished by their Cp distribution.

Table A.2: List of functions used in the benchmark set 2 (CEC 2005) with the corresponding global optima. Also, the bounds (or extant of the search space) in each direction is shown). In all problems, $z = x - o$, with o being the shifted optimum. For shifted rotated functions, $z = (x - o) \times M$, with M being a linear transformation matrix specified in [114]. A and B matrices are also specified in [114]. Note that functions f15 to f25 are composition functions that involve complex combinations of five or more functions and cannot be succinctly defined here.

No.	Name	Function	f*	Bounds
f1	Shifted Sphere Function	$\sum_{i=1}^{n} z_i^2 - 450$	-450	$[-100,100]$
f2	Shifted Schwefel's Function	$\sum_{i=1}^{n} \sum_{j=1}^{i} (z_j)^2 - 450$	-450	$[-100,100]$
f3	Shifted Rotated High Conditioned Elliptic Function	$\sum_{i=1}^{n} (10^6)^{\frac{i-1}{n-1}} - 450$	-450	$[-100,100]$
f4	Shifted Schwefel's Problem 1.2 with noise	$(\sum_{i=1}^{n} (\sum_{j=1}^{i})^2) * (1 + 0.4\lvert N(0,1)\rvert) - 450$	-450	$[-100,100]$
f5	Schwefel's Problem 2.6 with Global Optimum on Bounds	$\max(\lvert A_i x - B_i\rvert) - 310$	-310	$[-500,500]$
f6	Shifted Rosenbrock's Function	$\sum_{i=1}^{D-1} (100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + 390$	$+390$	$[-100,100]$
f7	Shifted Rotated Griewank's Function without Bounds	$\sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod i = 1 n cos(\frac{z_i}{\sqrt{i}}) + 1 + 180$	-180	$[0,600]$
f8	Shifted Rotated Ackley's Function with Global Optimum on Bounds	$-20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi z_i)) + 20 + e - 140$	-140	$[-32,32]$
f9	Shifted Rastrigin's Function	$\sum_{i=1}^{n} (z_i^2 - 10cos(2\pi z_i) + 10) - 330$	-330	$[-5,5]$
f10	Shifted Rotated Rastrigin's Function	$\sum_{i=1}^{n} (z_i^2 - 10cos(2\pi z_i) + 10) - 330$	-330	$[-5,5]$
f11	Shifted Rotated Weistrasss Function	$\sum_{i=1}^{n} (\sum_{k=1}^{kmax} [a^k cos(2\pi b^k (z_i + 0.5))]) - n\sum_{k=1}^{kmax} [a^k cos(2\pi b^k \cdot 0.5)] + 90$, $a = 0.5, b = 3, kmax = 20$	$+90$	$[-0.5,0.5]$
f12	Schwefel's Function 2.13	$\sum_{i=1}^{n} (A_i - B_i(x))^2 - 460$	-460	$[-\pi,\pi]$
f13	Expanded f8 plus f2	$f8(f2(x_1,x_2)) + f8(f2(x_2,x_3)) + \ldots + f8(f2(z_{n-1},z_n)) + f8(f2(z_D,z_1)) - 130$	-130	$[-3,1]$
f14	Shifted Rotated Expanded f6	$F(z_1,z_2) + F(z_2,z_3) + \ldots + F(z_{n-1},z_n) + F(z_n,z_1), F(x,y) = 0.5 + \frac{sin^2(\sqrt{x^2+z^2}-0.5)}{(1+0.001(x^2+y^2))^2}$	-300	$[-100,100]$
f15 - f25	See [114]	-	-	-

A.3 Appendices for Chapter 6

Appendix A - Indirect Method for Optimal Control

Trajectory optimization problems are generally expressed in the form of minimizing an integral (Eq.A.16) subjected to dynamics constraints (Eq.A.16), initial value of states (Eq.A.17), and a terminal cost constraint (Eq.A.18, L is the path cost, x is an n-dimensional state vector, u is an m-dimensional control vector and ϕ is a p-dimensional terminal constraint vector. [191]

$$J = \Phi(t_f, x(t_f)) + \int_{t_0}^{t_f} L(t, x(t), u(t))dt \tag{A.16}$$

$$\dot{x} = f(t, x(t), u(t)) \tag{A.17}$$

$$x(t_0) = x_0 \tag{A.18}$$

$$\phi(t_f, x(t_f)) = 0 \tag{A.19}$$

The objective in this trajectory optimization problem is to minimize time of flight, t_f. Indirect methods involve satisfying the necessary conditions of optimality using Euler-Lagrange equations given by Eq.A.20 - A.21. λ in the following equations is an n-dimensional co-state vector and H is the Hamiltonian.

$$\dot{\lambda} = -\frac{dH}{dx} \tag{A.20}$$

$$\frac{dH}{du^*} = 0 \tag{A.21}$$

where

$$\mathrm{H} = \mathrm{L} + \lambda^{\mathrm{T}}\dot{x}$$

The Hamiltonian for the given problem and the time derivatives of co-states were then found using Eq.A.20 and are described by Eq.A.22 - A.27.

$$\dot{\lambda_x} = 0 \tag{A.22}$$

$$\dot{\lambda_y} = 0 \tag{A.23}$$

$$\dot{\lambda_z} = 0 \tag{A.24}$$

$$\dot{\lambda_v} = -\lambda_x cos\gamma cos\psi - \lambda_y cos\gamma sin\psi - \lambda_z sin\gamma + \frac{\lambda_\psi \sqrt{1-u^2}\,(L+Tsin\alpha)}{mv^2 cos\gamma} - \lambda_\gamma \left(\frac{gcos\gamma}{v^2} - \frac{u(L+Tsin\alpha)}{mv^2} \right) \tag{A.25}$$

$$\dot{\lambda_\psi} = vcos\gamma(\lambda_x sin\psi - \lambda_y cos\psi) \tag{A.26}$$

$$\dot{\lambda_\gamma} = \lambda_x vcos(\psi)sin\gamma + \lambda_y vsin\gamma sin\psi - \lambda_z vcos\gamma - \lambda_\gamma \left(\frac{gsin\gamma}{v} - \frac{gucos\gamma(D+mgsin\gamma)}{Tvsin\alpha} \right) - \lambda_\psi \left(\frac{sin\gamma\sqrt{1-u^2}(L+Tsin\alpha)}{mvcos\gamma^2} - \frac{g\sqrt{1-u^2}(D+mgsin\gamma)}{Tvsin\alpha} \right) \tag{A.27}$$

The control laws were found using Eq.A.21 and are described by Eq.A.28 - A.29.

$$u_1 = -\frac{\lambda_\gamma}{\sqrt{\lambda_\gamma^2 + (\lambda_\psi sec\gamma)^2}} \tag{A.28}$$

$$u_2 = \frac{\lambda_\gamma}{\sqrt{\lambda_\gamma^2 + (\lambda_\psi sec\gamma)^2}} \tag{A.29}$$

It is to be noted that two control law options were found. Using Pontryagin's minimum principle, the control law that minimizes the Hamiltonian was then chosen at each data point. [191]

Appendix B - Engine Analysis block

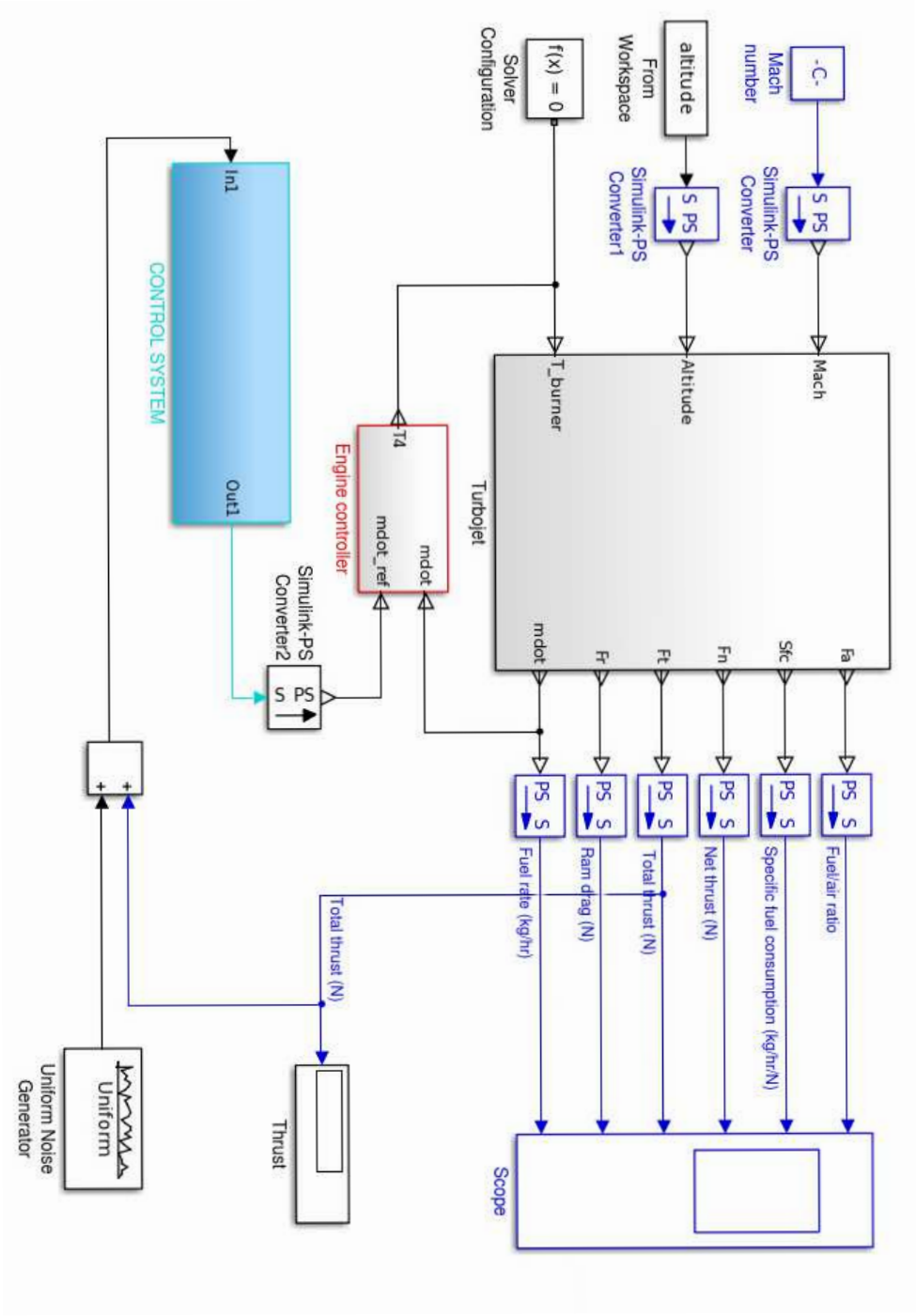


Figure A.6.: Simulink model of the turbojet Engine used for propulsion system analysis

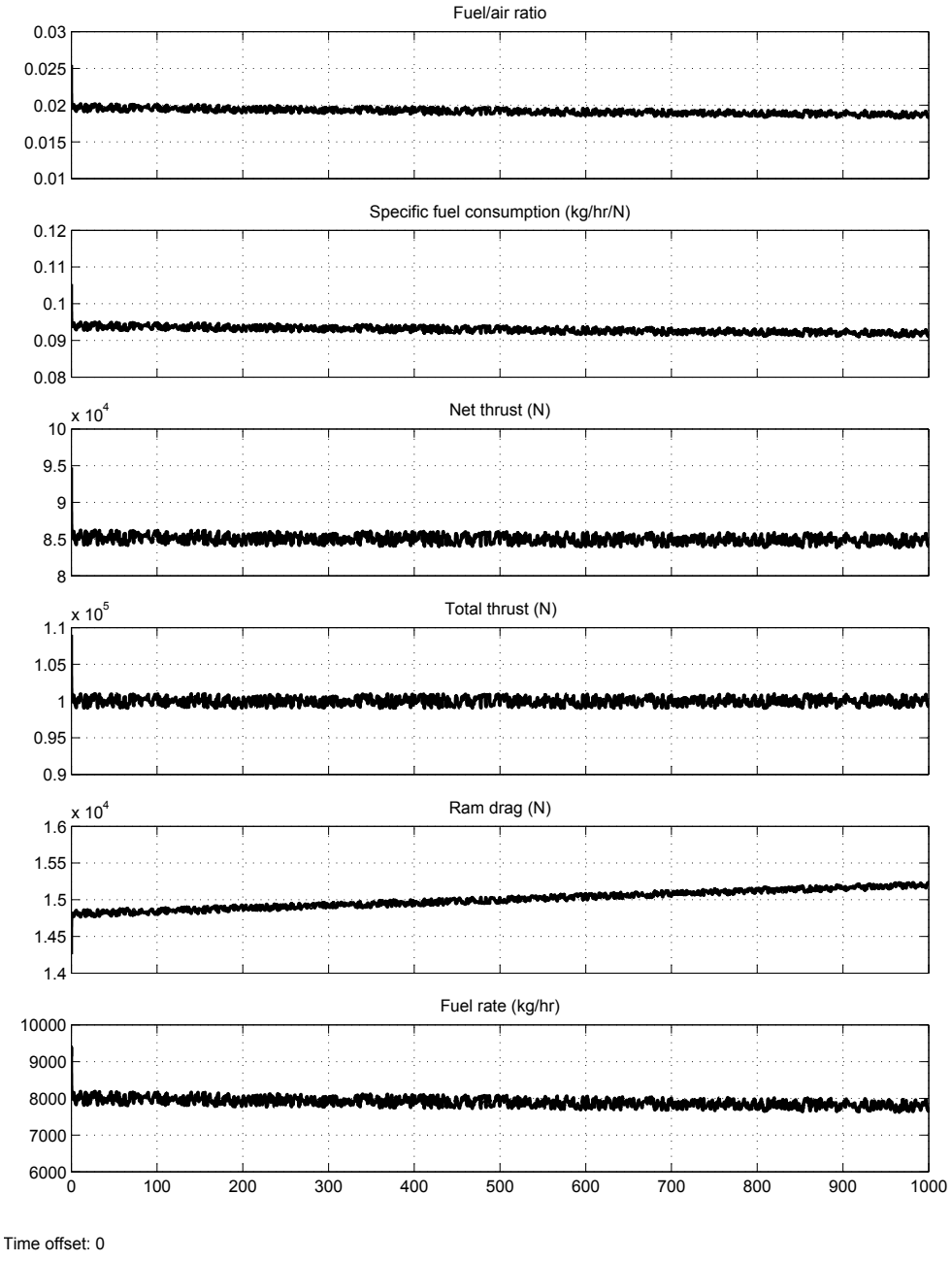


Figure A.7.: Outputs generated by the Simulink model of the turbojet Engine with the aim of maintaining a constant total thrust along the trajectory. Note that the total time of the trajectory is divide into 1000 time steps.

Appendix C - Parallel Runway Airport Map

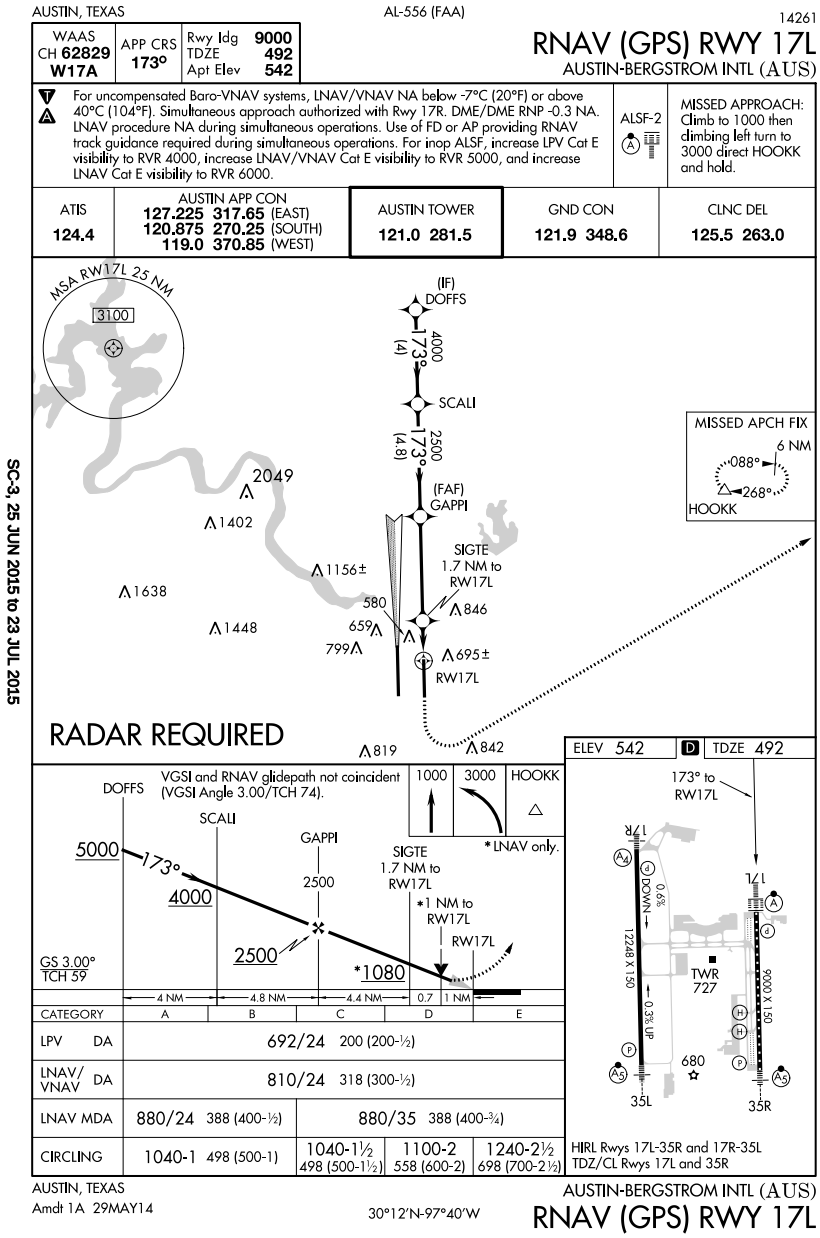


Figure A.8.: Final approach to Runway 17L, Austin Bergstrom International airport

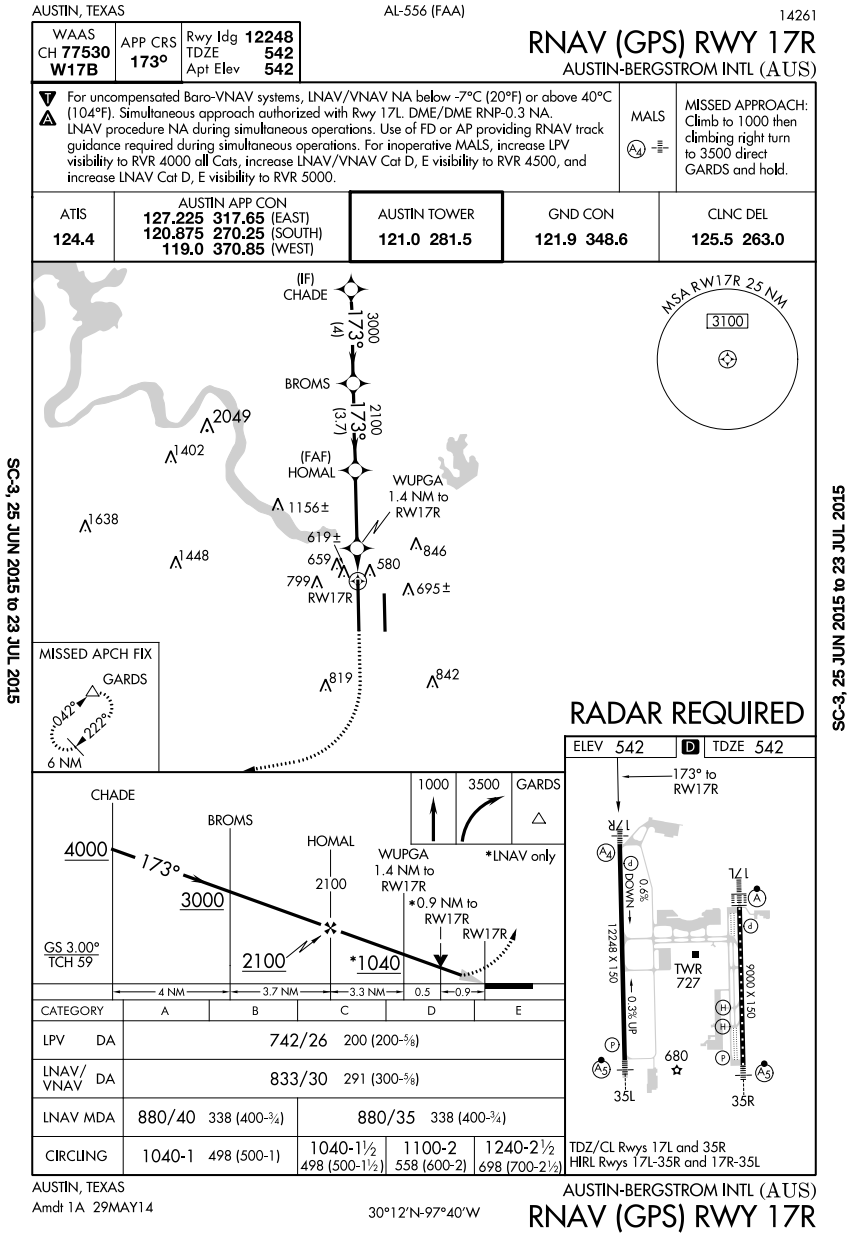


Figure A.9.: Final approach to Runway 17R, Austin Bergstrom International airport

LIST OF REFERENCES

LIST OF REFERENCES

[1] AA Giunta, O Golividov, DL Knill, B Grossman, WH Mason, LT Watson, and RT Haftka. Multidisciplinary design optimization of advanced aircraft configurations. In *Fifteenth International Conference on Numerical Methods in Fluid Dynamics*, pages 14–34. Springer, 1997.

[2] JJ Alonso, P LeGresley, and V Pereyra. Aircraft design optimization. *Mathematics and Computers in Simulation*, 79(6):1948–1958, 2009.

[3] Thierry Lefebvre, Peter Schmollgruber, Christophe Blondeau, and Gérald Carrier. Aircraft conceptual design in a multi-level, multi-fidelity, multidisciplinary optimization process. In *28th International Congress of the Aeronautical Sciences*, pages 1–11, 2012.

[4] Marian Nemec, David W Zingg, and Thomas H Pulliam. Multipoint and multi-objective aerodynamic shape optimization. *AIAA journal*, 42(6):1057–1065, 2004.

[5] RP Henderson, JRRA Martins, and RE Perez. Aircraft conceptual design for optimal environmental performance. *Aeronautical Journal*, 116(1175):1, 2012.

[6] TD Robinson, KE Willcox, MS Eldred, and R Haimes. Multifidelity optimization for variablecomplexity design. In *Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Portsmouth, VA*, 2006.

[7] Sricharan Kishore Ayyalasomayajula. A formulation to analyze system-of-systems problems: A case study of airport metroplex operations. 2011.

[8] Paul Kostekm and Pierre DeChazelles. Framework for the application of systems engineering in the commercial aircraft domain. 2000.

[9] Robert A Wolf. Multiobjective collaborative optimization of systems of systems. Technical report, DTIC Document, 2005.

[10] Muharrem Mane, William A Crossley, and Antonius Nusawardhana. System-of-systems inspired aircraft sizing and airline resource allocation via decomposition. *Journal of Aircraft*, 44(4):1222–1235, 2007.

[11] Christine Taylor and Olivier L de Weck. Integrated transportation network design optimization. *AIAA Paper*, 1912, 2006.

[12] A Nusawardhana and William Crossley. Allocating variable resources over a finite time horizon to combine aircraft sizing and airline planning. 2005.

[13] Harrison M Kim and I Jessica Hidalgo. System of systems optimization by pseudo-hierarchical multistage model. In *Proceedings 11th AIAA/ISSMO multidisciplinary analysis and optimization conference*, 2006.

[14] Navindran Davendralingam and William Crossley. Robust optimization of aircraft design and airline network design incorporating economic trends. 2011.

[15] Navindran Davendralingam and Daniel DeLaurentis. A robust optimization framework to architecting system of systems. *Procedia Computer Science*, 16:255–264, 2013.

[16] Ferat Sahin, Prasanna Sridhar, Ben Horan, Vikraman Raghavan, and Mo Jamshidi. System of systems approach to threat detection and integration of heterogeneous independently operable systems. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 1376–1381. IEEE, 2007.

[17] James Allison, Michael Kokkolaras, Marc Zawislak, and Panos Y Papalambros. On the use of analytical target cascading and collaborative optimization for complex system design. In *6th World Congress on Structural and Multidisciplinary Optimization*, 2005.

[18] John E Dennis Jr, Sharon F Arroyo, Evin J Cramer, and Paul D Frank. Problem formulations for systems of systems. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 64–71. IEEE, 2005.

[19] Gautam Marwaha and Michael Kokkolaras. System-of-systems approach to air transportation design using nested optimization and direct search. *Structural and Multidisciplinary Optimization*, pages 1–17.

[20] Jaroslaw Sobieszczanski-Sobieski. Integrated system-of-systems synthesis (isss). *AIAA Journal*, 2006.

[21] Hyung Min Kim, D Geoff Rideout, Panos Y Papalambros, and Jeffrey L Stein. Analytical target cascading in automotive vehicle design. *Journal of Mechanical Design*, 125(3):481–489, 2003.

[22] András Sóbester, Andy J Keane, James Scanlan, and Neil W Bressloff. Conceptual design of uav airframes using a generic geometry service. In *Infotech@ Aerospace Conferences. American Institute of Aeronautics and Astronautics*, 2005.

[23] GJ Grenzdörffer, A Engel, and B Teichert. The photogrammetric potential of low-cost uavs in forestry and agriculture. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 31(B3):1207–1214, 2008.

[24] Stanley R Herwitz, Lee F Johnson, JC Arvesen, R Higgins, J Leung, and S Dunagan. Precision agriculture as a commercial application for solar-powered unmanned aerial vehicles. In *1st American Institute of Aeronautics and Astronautics UAV Conference*, 2002.

[25] Jacopo Primicerio, Salvatore Filippo Di Gennaro, Edoardo Fiorillo, Lorenzo Genesio, Emanuele Lugato, Alessandro Matese, and Francesco Primo Vaccari. A flexible unmanned aerial vehicle for precision agriculture. *Precision Agriculture*, 13(4):517–523, 2012.

[26] Mike Karst. Uavs in agriculture: Rules of the sky, December 2013.

[27] Civil Aviation Authority. Managing aviation noise. Technical report, 2014.

[28] Andrew Sirotnik. Amid amazon's drones and dominance, what's a retailer to do?, December 2013.

[29] Doug Gross. Amazon's drone delivery: How would it work?, December 2013.

[30] Jaroslaw Sobieszczanski-Sobieski. Integrated system-of-systems synthesis. *AIAA journal*, 46(5):1072–1080, 2008.

[31] Braun RD and Kroo IM. Development and application of the collaborative optimization architecture in a multidisciplinary design environment. 1995.

[32] HW Chi and CL Bloebaum. Concurrent subspace optimization for mixed-variable coupled engineering systems. In *Proceedings of the 6th AIAA/AFOSR/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA*, 1996.

[33] Muharrem Mane, William A Crossley, and Antonius Nusawardhana. System-of-systems inspired aircraft sizing and airline resource allocation via decomposition. *Journal of Aircraft*, 44(4):1222–1235, 2007.

[34] John E Dennis Jr, Sharon F Arroyo, Evin J Cramer, and Paul D Frank. Problem formulations for systems of systems. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 64–71. IEEE, 2005.

[35] Sharon F Arroyo, Evin J Cramer, John E Dennis Jr, and Paul D Frank. Comparing problem formulations for coupled sets of components. *Optimization and Engineering*, 10(4):557–573, 2009.

[36] Alberto Sangiovanni-Vincentelli. Quo vadis, sld? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[37] Sandor Becz, A Pinto, LE Zeidner, A Banaszuk, R Khire, and HM Reeve. Design system for managing complexity in aerospace systems. In *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, 2010.

[38] Nikolaos Papakonstantinou, Seppo Sierla, David C Jensen, and Irem Y Tumer. Simulation of interactions and emergent failure behavior during complex system design. *Journal of Computing and Information Science in Engineering*, 12:031007, 2012.

[39] Tolga Kurtoglu, Irem Y Tumer, and David C Jensen. A functional failure reasoning methodology for evaluation of conceptual system architectures. *Research in Engineering Design*, 21(4):209–234, 2010.

[40] Daniel DeLaurentis, Muharrem Mane, and Navindran Davendralingam. Capability and development risk management in system-of-systems architectures: A portfolio approach to decision-making, 2012.

[41] Navindran Davendralingam and Daniel DeLaurentis. A robust optimization framework to architecting system of systems. *Procedia Computer Science*, 16:255–264, 2013.

[42] Roxanne A Moore, David A Romero, and Christiaan JJ Paredis. Value-based global optimization. *Journal of Mechanical Design*, 136(4):041003, 2014.

[43] Theresa Dawn Robinson. *Surrogate-based optimization using multifidelity models with variable parameterization*. PhD thesis, Massachusetts Institute of Technology, 2007.

[44] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[45] Shreyas Vathul Subramanian and Daniel A DeLaurentis. A hybrid differential evolution self-organizing-map algorithm for optimization of expensive black-box functions. In *AIAA Aviation - 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2014.

[46] Nils Bertschinger, David H Wolpert, Eckehard Olbrich, and Juergen Jost. Information geometry of noncooperative games. *arXiv preprint arXiv:1401.0001*, 2013.

[47] Jonathan ME Gabbai. *Complexity and the aerospace industry: Understanding emergence by relating structure to performance using multi-agent systems*. PhD thesis, Citeseer, 2005.

[48] Mark V Arena, Obaid Younossi, Kevin Brancato, Irv Blickstein, and Clifford A Grammich. Why has the cost of fixed-wing aircraft risen? a macroscopic examination of the trends in us military aircraft costs over the past several decades. Technical report, DTIC Document, 2008.

[49] Jean M Carlson and John Doyle. Complexity and robustness. *Proceedings of the National Academy of Sciences of the United States of America*, 99(Suppl 1):2538–2545, 2002.

[50] Joshua D Summers and Jami J Shah. Mechanical engineering design complexity metrics: size, coupling, and solvability. *Journal of Mechanical Design*, 132:021004, 2010.

[51] Doug Stuart, Raju Mattikalli, Daniel DeLaurentis, and Jami Shah. Complexity and adaptability, 2012.

[52] Dan Braha and Oded Maimon. The measurement of a design structural and functional complexity. In *A Mathematical Theory of Design: Foundations, Algorithms and Applications*, pages 241–277. Springer, 1998.

[53] Michael Lamport Commons. Introduction to the model of hierarchical complexity and its relationship to postformal action. *World Futures*, 64(5-7):305–320, 2008.

[54] Robert A Wolf. Multiobjective collaborative optimization of systems of systems. Technical report, DTIC Document, 2005.

[55] Robert A Wolf. Multiobjective collaborative optimization of systems of systems. Technical report, DTIC Document, 2005.

[56] Harrison M Kim and I Jessica Hidalgo. System of systems optimization by pseudo-hierarchical multistage model. In *11th AIAA/ISSMO mutlidisplinary analysis and optimization conference*, 2006.

[57] Sricharan Ayyalasomayajula, Donald N Fry, and Daniel A DeLaurentis. A framework for formulating design problems in a system-of-systems context. 2008.

[58] Muharrem Mane, William A Crossley, and Antonius Nusawardhana. System-of-systems inspired aircraft sizing and airline resource allocation via decomposition. *Journal of Aircraft*, 44(4):1222–1235, 2007.

[59] Christine Taylor and Olivier L de Weck. Integrated transportation network design optimization. 2006.

[60] A Nusawardhana and William Crossley. Allocating variable resources over a finite time horizon to combine aircraft sizing and airline planning.

[61] Joshua B Frommer. System of systems design: Evaluating aircraft in a fleet context using reliability and non-deterministic approaches. 2008.

[62] Shreyas Vathul Subramanian and Daniel A. DeLaurentis. Hierarchical complexity guided optimization of system-of-systems with evolving design spaces. In *4th International Engineering Systems Symposium (CESUN 2014)*. Stevens Institute of Technology, 2014.

[63] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33):3190–3218, 2007.

[64] Ferat Sahin, Mo Jamshidi, and Prassana Sridhar. A discrete event xml based simulation framework for system of systems architectures. In *System of Systems Engineering, 2007. SoSE'07. IEEE International Conference on*, pages 1–7. IEEE, 2007.

[65] Peng-cheng LUO, Pan-feng FU, and Jing-lun ZHOU. Framework to evaluate the combat capability of weapons sos [j]. *Systems Engineering and Electronics*, 1:72–75, 2005.

[66] Jing-yu YANG, Guang-ya SI, and Xiao-feng HU. Study on simulation-based exploratory analysis method of information warfare system of system (sos) encounter [j]. *Acta Simulata Systematica Sinica*, 6, 2005.

[67] Daniel A DeLaurentis, William A Crossley, and Muharrem Mane. Taxonomy to guide systems-of-systems decision-making in air transportation problems. *Journal of Aircraft*, 48(3):760–770, 2011.

[68] In Gwun Jang and Byung Man Kwak. Evolutionary topology optimization using design space adjustment based on fixed grid. *International journal for numerical methods in engineering*, 66(11):1817–1840, 2006.

[69] Il Yong Kim and Byung Man Kwak. Design space optimization using a numerical design continuation method. *International Journal for Numerical Methods in Engineering*, 53(8):1979–2002, 2002.

[70] In Gwun Jang and Byung Man Kwak. Design space optimization using design space adjustment and refinement. *Structural and Multidisciplinary optimization*, 35(1):41–54, 2008.

[71] Mohammad Kashif Zahir and Zhenghong Gao. Variable-fidelity optimization with design space reduction. *Chinese Journal of Aeronautics*, 26(4):841–849, 2013.

[72] R Storn and K Price. Differential evolution : A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[73] Jakob Vesterstrom and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. IEEE Congress on*, volume 2, pages 1980–1987, 2004.

[74] Hideyuki Takagi and Denis Pallez. Paired comparison-based interactive differential evolution. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 475–480. IEEE, 2009.

[75] Pinar Civicioglu and Erkan Besdok. A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review*, pages 1–32, 2013.

[76] K.V. Price. Differential evolution: a fast and simple numerical optimizer. In *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, pages 524–527, 1996.

[77] N. Mezura-Montes, Jesus Velazquez-Reyes, and Carlos A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485–492. ACM, 2006.

[78] Teuvo Kohonen. *Self-organizing maps*, volume 30. Springer, 2001.

[79] Hujun Yin. The self-organizing maps: Background, theories, extensions and applications. In *Computational intelligence: a compendium*, pages 715–762. Springer, 2008.

[80] Xiang Xiao, Ernst Dow, Russell Eberhart, Zina Ben Miled, and Robert Oppelt. A hybrid self-organizing maps and particle swarm optimization approach. *Concurrency and Computation: Practice and Experience*, 16:895–915, 2004.

[81] Tiago P.F. de Lima, Adenilton J da Silva, and Teresa B Ludermir. Selection and fusion of neural networks via differential evolution. In *Advances in Artificial Intelligence-IBERAMIA 2012*, pages 149–158. Springer, 2012.

[82] P. Julrode and S. Supratid. Investigation of using variant differential evolutions on optimizing 2-level self-organizing map. In *Computer Science and Software Engineering (JCSSE), 2011 Eighth International Joint Conference on*, pages 123–127, 2011.

[83] Su Miin-Tsair, Chen Cheng-Hung, Lin Cheng-Jian, and Lin Chin-Teng. A rule-based symbiotic {MOdified} differential evolution for self-organizing neuro-fuzzy systems. *Applied Soft Computing*, 11:4847 – 4858, 2011.

[84] Xiang Xiao, Ernst Dow, Russell Eberhart, Zina Ben Miled, and Robert Oppelt. Selection and fusion of neural networks via differential evolution. *Advances in Artificial Intelligence*, 7637:149–158, 2012.

[85] Shigeru Obayashi and Daisuke Sasaki. Visualization and data mining of pareto solutions using self-organizing map. In *Evolutionary multi-criterion optimization*, pages 796–809. Springer, 2003.

[86] Jon Louis Bentley, Franco P Preparata, and Mark G Faust. Approximation algorithms for convex hulls. *Communications of the ACM*, 25:64–68, 1982.

[87] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22:469–483, 1996.

[88] Murray Gerstenhaber. Theory of convex polyhedral cones. *Activity analysis of production and allocation*, pages 298–316, 1951.

[89] Janez Brest, Saso Greiner, Borko Bo skovic, Marjan Mernik, and Zumer Viljem. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10:646–657, 1951.

[90] JJ Liang, PN Suganthan, and K Deb. Novel composition test functions for numerical global optimization. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 68–75, 2005.

[91] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[92] Quan-Ke Pan, Ponnuthurai N Suganthan, Ling Wang, Liang Gao, and Rammohan Mallipeddi. A differential evolution algorithm with self-adapting strategy and control parameters. *Computers & Operations Research*, 38(1):394–408, 2011.

[93] Slawomir Koziel and Leifur Leifsson. Surrogate-based aerodynamic shape optimization by variable-resolution models. *AIAA Journal*, 51(1):94–106, 2012.

[94] Howard P Buckley and David W Zingg. Approach to aerodynamic design through numerical optimization. *AIAA Journal*, pages 1–10, 2013.

[95] Oleg Chernukhin and David W Zingg. Multimodality and global optimization in aerodynamic design. *AIAA Journal*, 51(6):1342–1354, 2013.

[96] Mark Drela. Design and optimization method for multi-element airfoils. *AIAA paper*, pages 93–0969, 1993.

[97] Luc Huyse and R Michael Lewis. *Aerodynamic shape optimization of two-dimensional airfoils under uncertain conditions*. Citeseer, 2001.

[98] Alessandro Vicini and Domenico Quagliarella. Airfoil and wing design through hybrid optimization strategies. *AIAA journal*, 37(5):634–641, 1999.

[99] Jamshid A Samareh. Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. *AIAA journal*, 39(5):877–884, 2001.

[100] Domenico Quagliarella and Antonio Della Cioppa. Genetic algorithms applied to the aerodynamic design of transonic airfoils. *Journal of Aircraft*, 32(4):889–891, 1995.

[101] Brenda M Kulfan and John E Bussoletti. Fundamental parametric geometry representations for aircraft component shapes. 2006.

[102] Kevin A Lane and David D Marshall. Inverse airfoil design utilizing cst parameterization. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, pages 1228–1242, 2010.

[103] Onay Urfalioglu and Orhan Arikan. Self-adaptive randomized and rank-based differential evolution for multimodal problems. *Journal of Global Optimization*, 51(4):607–640, 2011.

[104] Haiyan Lu and Weiqi Chen. Self-adaptive velocity particle swarm optimization for solving constrained optimization problems. *Journal of Global Optimization*, 41(3):427–445, 2008.

[105] Amin Nobakhti and Hong Wang. A simple self-adaptive Differential Evolution algorithm with application on the ALSTOM gasifier. *Applied Soft Computing*, 8:350–370, 2008.

[106] Rammohan Mallipeddi, Ponnuthurai N. Suganthan, Quan-Ke Pan, and Mehmet Fatih Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11:1679–1696, 2011.

[107] Tetsuyuki Takahama and Setsuko Sakai. Reducing function evaluations using adaptively controlled differential evolution with rough approximation model. In *computational intelligence in expensive optimization problems*, pages 111–129. Springer, 2010.

[108] Richard D. Lawrence, George S. Almasi, and Holly E. Rushmeier. A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems. *Data Mining and Knowledge Discovery*, 3(2):171–195, 1999.

[109] Michael J Todd and E Alper Yıldırım. On khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discrete Applied Mathematics*, 155(13):1731–1744, 2007.

[110] Peng Sun and Robert M Freund. Computation of minimum-volume covering ellipsoids. *Operations Research*, 52(5):690–706, 2004.

[111] JJ Liang, PN Suganthan, and K Deb. Novel composition test functions for numerical global optimization. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 68–75. IEEE, 2005.

[112] Nikolaus Hansen. Compilation of results on the 2005 cec benchmark function set. *Online, May*, 2006.

[113] Christian L Müller and Ivo F Sbalzarini. Global characterization of the cec 2005 fitness landscapes using fitness-distance analysis. In *Applications of Evolutionary Computation*, pages 294–303. Springer, 2011.

[114] Ponnuthurai N Suganthan, Nikolaus Hansen, Jing J Liang, Kalyanmoy Deb, Y-Po Chen, Anne Auger, and S Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. 2005.

[115] Jani Ronkkonen, Saku Kukkonen, and Kenneth V Price. Real-parameter optimization with differential evolution. In *Evolutionary Computation, 2005. CEC2005. IEEE Congress on*. IEEE, 2005.

[116] Lam T. Bui, Yin Shan, Qi Fan, and Hussein A. Abbass. Comparing two versions of differential evolution in real parameter optimization. In *Evolutionary Computation, 2005. CEC2005. IEEE Congress on*. IEEE, 2005.

[117] M. Fatih Tasgetiren, Yun-Chia Liang, and Gunes Gencyilmaz. A differential evolution algorithm for continuous function optimization. In *Evolutionary Computation, 2005. CEC2005. IEEE Congress on*. IEEE, 2005.

[118] Dimitri P Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010:1–38, 2011.

[119] Dimitri P Bertsekas. Incremental proximal methods for large scale convex optimization. *Mathematical programming*, 129(2):163–195, 2011.

[120] Angelia Nedic. Random projection algorithms for convex set intersection problems. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 7655–7660. IEEE, 2010.

[121] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, 2009.

[122] Michael G Rabbat and Robert D Nowak. Quantized incremental algorithms for distributed optimization. *Selected Areas in Communications, IEEE Journal on*, 23(4):798–808, 2005.

[123] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *Automatic Control, IEEE Transactions on*, 57(3):592–606, 2012.

[124] Björn Johansson, Maben Rabi, and Mikael Johansson. A randomized incremental subgradient method for distributed optimization in networked systems. *SIAM Journal on Optimization*, 20(3):1157–1170, 2009.

[125] S Sundhar Ram, A Nedić, and Venugopal V Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147(3):516–545, 2010.

[126] Yair Censor, Alvaro R De Pierro, and Maroun Zaknoon. Steered sequential projections for the inconsistent convex feasibility problem. *Nonlinear Analysis: Theory, Methods & Applications*, 59(3):385–405, 2004.

[127] Yair Censor, Avi Motova, and Alexander Segal. Perturbed projections and subgradient projections for the multiple-sets split feasibility problem. *Journal of Mathematical Analysis and Applications*, 327(2):1244–1256, 2007.

[128] Charles Byrne. Bregman-legendre multidistance projection algorithms for convex feasibility and optimization. *Studies in Computational Mathematics*, 8:87–99, 2001.

[129] Ron Aharoni, Abraham Berman, and Yair Censor. An interior points algorithm for the convex feasibility problem. *Advances in Applied Mathematics*, 4(4):479–489, 1983.

[130] John W Chinneck. *Feasibility and Infeasibility in Optimization:: Algorithms and Computational Methods*, volume 118. Springer Science & Business Media, 2007.

[131] Edoardo Amaldi, Marc E Pfetsch, and Leslie E Trotter Jr. Some structural and algorithmic properties of the maximum feasible subsystem problem. In *Integer Programming and Combinatorial Optimization*, pages 45–59. Springer, 1999.

[132] Marc E Pfetsch. *Branch-and-cut for the maximum feasible subsystem problem*. Konrad-Zuse-Zentrum für Informationstechnik, 2005.

[133] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proceedings of 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 93, 1993.

[134] Heinz H Bauschke and Jonathan M Borwein. On projection algorithms for solving convex feasibility problems. *SIAM review*, 38(3):367–426, 1996.

[135] Mengdi Wang and Dimitri P Bertsekas. Incremental constraint projection-proximal methods for nonsmooth convex optimization. Technical report, Technical report, MIT, 2013.

[136] Frank Deutsch and Hein Hundal. The rate of convergence for the cyclic projections algorithm iii: regularity of convex sets. *Journal of Approximation Theory*, 155(2):155–184, 2008.

[137] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods and applications*. Springer Science & Business Media, 2003.

[138] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, 2001.

[139] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

[140] SF Rahmatalla and CC Swan. A q4/q4 continuum structural topology optimization implementation. *Structural and Multidisciplinary Optimization*, 27(1-2):130–135, 2004.

[141] Waqas Saleem, M Aurangzeb Khan, and Sajid Raza Ch. Formulation and execution of structural topology optimization for practical design solutions. *Journal of Optimization Theory and Applications*, 152(2):517–536, 2012.

[142] Ronald HW Hoppe, Svetozara I Petrova, and V Schulz. Primal-dual newton-type interior-point method for topology optimization. *Journal of Optimization Theory and Applications*, 114(3):545–571, 2002.

[143] Alfred O Hero and Douglas Cochran. Sensor management: Past, present, and future. *Sensors Journal, IEEE*, 11(12):3064–3075, 2011.

[144] Federico S Cattivelli, Cassio G Lopes, and Ali H Sayed. Diffusion recursive least-squares for distributed estimation over adaptive networks. *Signal Processing, IEEE Transactions on*, 56(5):1865–1877, 2008.

[145] Vassilis Kekatos and Georgios B Giannakis. Distributed robust power system state estimation. *Power Systems, IEEE Transactions on*, 28(2):1617–1626, 2013.

[146] Michael Rabbat and Robert Nowak. Distributed optimization in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 20–27. ACM, 2004.

[147] Peter J Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[148] Jean-Benoist Léger and Michel Kieffer. Guaranteed robust distributed estimation in a network of sensors. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 3378–3381. IEEE, 2010.

[149] Véronique Delouille, Ramesh Neelamani, and Richard Baraniuk. Robust distributed estimation in sensor networks using the embedded polygons algorithm. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 405–413. ACM, 2004.

[150] David Moore, John Leonard, Daniela Rus, and Seth Teller. Robust distributed network localization with noisy range measurements. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 50–61. ACM, 2004.

[151] Q Li and WS Wong. Optimal estimator for distributed anonymous observers. *Journal of optimization theory and applications*, 140(1):55–75, 2009.

[152] Lin Xiao, Stephen Boyd, and Sanjay Lall. A scheme for robust distributed sensor fusion based on average consensus. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 63–70. IEEE, 2005.

[153] Maziar S Hemati, Jeff D Eldredge, and Jason L Speyer. Wake sensing for aircraft formation flight. *Journal of Guidance, Control, and Dynamics*, 37(2):513–524, 2014.

[154] J Slotnick, Roger W Clark, Douglas M Friedman, Yoram Yadlin, David T Yeh, John E Carr, Michael J Czech, and Stefan W Bieniawski. Computational aerodynamic analysis for the formation flight for aerodynamic benefit program. In *AIAA Science and Technology Forum and Exposition*, 2014.

[155] David J Halaas, Stefan R Bieniawski, Brian T Whitehead, Tristan Flanzer, and William B Blake. Formation flight for aerodynamic benefit simulation development and validation. In *AIAA Science and Technology Forum and Exposition*, 2014.

[156] John-Paul B Clarke, Nhut T Ho, Liling Ren, John A Brown, Kevin R Elmer, Katherine Zou, Christopher Hunting, Daniel L McGregor, Belur N Shivashankara, Kwok-On Tong, et al. Continuous descent approach: Design and flight test for louisville international airport. *Journal of Aircraft*, 41(5):1054–1066, 2004.

[157] Jian Yang, Zhihua Qu, Jing Wang, Richard Hull, et al. A real-time optimized path planning for a fixed wing vehicle flying in a dynamic and uncertain environment. In *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, pages 96–102. IEEE, 2005.

[158] David G Hull. Conversion of optimal control problems into parameter optimization problems. *Journal of Guidance, Control, and Dynamics*, 20(1):57–60, 1997.

[159] Frederick N Fritsch and Ralph E Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980.

[160] F Gembicki and Yacov Y Haimes. Approach to performance and sensitivity multiobjective optimization: The goal attainment method. *Automatic Control, IEEE Transactions on*, 20(6):769–771, 1975.

[161] Shih-Ping Han. A globally convergent method for nonlinear programming. *Journal of optimization theory and applications*, 22(3):297–309, 1977.

[162] Michael JD Powell. A fast algorithm for nonlinearly constrained optimization calculations. In *Numerical analysis*, pages 144–157. Springer, 1978.

[163] Nicolas E Antoine and Ilan M Kroo. Optimizing aircraft and operations for minimum noise. In *AIAA's Aircraft Technology, Integration, and Operations (ATIO)*. AIAA, 2002.

[164] David P Lockard and Geoffrey M Lilley. The airframe noise reduction challenge. 2004.

[165] Leifur Thor Leifsson. Multidisciplinary design optimization of low-noise transport aircraft, phd thesis. 2005.

[166] Ronald P King and Jeffrey R Davis. Community noise: health effects and management. *International journal of hygiene and environmental health*, 206(2):123–131, 2003.

[167] Ian Jopson, Darren Rhodes, and Peter Havelock. Aircraft noise model validation : how accurate do we need to be. Civil Aviation Authority, 2002.

[168] Civil Aviation Authority. Noise data for the first 17 months of boeing 787 operations at heathrow airport. Technical report, 2014.

[169] JB Ollerhead. *The CAA Aircraft Noise Contour Model: ANCON Version 1*. Civil Aviation Authority, 1992.

[170] Andrew Ning, Tristan C Flanzer, and Ilan M Kroo. Aerodynamic performance of extended formation flight. *Journal of aircraft*, 48(3):855–865, 2011.

[171] Coleman duP Donaldson, Richard S Snedeker, and Roger D Sullivan. Calculation of aircraft wake velocity profiles and comparisonwith experimental measurements. *Journal of Aircraft*, 11(9):547–555, 1974.

[172] Leo J Garodz. Measurements of boeing 747, lockheed c5a and other aircraft vortex wake characteristics by tower fly-by technique. In *Aircraft Wake Turbulence and Its Detection*, pages 265–285. Springer, 1971.

[173] Leo J Garodz, David Lawrence, and Nelson Miller. The measurement of the boeing 727 trailing vortex system using the tower fly-by technique. Technical report, DTIC Document, 1974.

[174] Kethryn M Butler. Estimation of wake vortex advection and decay using meteorological sensors and aircraft data. Technical report, DTIC Document, 1993.

[175] Philip M Gresho. Some current cfd issues relevant to the incompressible navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 87(2):201–252, 1991.

[176] John Cornthwaite. Pressure poisson method for the incompressible navier-stokes equations using galerkin finite elements. 2013.

[177] Suhas V Patankar and D Brian Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15(10):1787–1806, 1972.

[178] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 2001.

[179] Joaquim RRA Martins and Andrew B Lambe. Multidisciplinary design optimization: a survey of architectures. *AIAA journal*, 51(9):2049–2075, 2013.

[180] Brian Roth and Ilan Kroo. Enhanced collaborative optimization. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Victoria, BC*, 2008.

[181] Shreyas Vathul Subramanian and Daniel A DeLaurentis. Hierarchical complexity guided optimization of systems-of-systems with evolving design spaces. *4th CESUN, June*, 2014.

[182] Mark Drela. Xfoil: An analysis and design system for low reynolds number airfoils. In *Low Reynolds number aerodynamics*, pages 1–12. Springer, 1989.

[183] Mark Drela. Pros and cons of airfoil optimization. *Frontiers of Computational Fluid dynamics*, pages 363–381, 1998.

[184] Mark Guiles. A sequential, multi-complexity topology optimization process for aero-elastic wing structure design. Master's thesis, Purdue University, 12 2011.

[185] Dimitris Bertsimas, David B Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM review*, 53(3):464–501, 2011.

[186] Vipin B Gawande, AS Dhoble, and DB Zodpe. Cfd analysis to study effect of circular vortex generator placed in inlet section to investigate heat transfer aspects of solar air heater. *The Scientific World Journal*, 2014, 2014.

[187] Kenrick A Waithe. Source term model for vortex generator vanes in a navier-stokes computer code. In *42nd AIAA Aerospace Sciences Meeting and Exhibit*. AIAA, 2004.

[188] Manton Lane. A new vortex generator model for use in complex configuration cfd solvers. 2001.

[189] Sambarta Dasgupta, Swagatam Das, Arijit Biswas, and Ajith Abraham. On stability and convergence of the population-dynamics in differential evolution. *Ai Communications*, 22:1–20, 2009.

[190] Daniela Zaharie. Critical values for the control parameters of differential evolution algorithms. In *Proceedings of MENDEL*, pages 62–67, 2002.

[191] James M Longuski, José J Guzmán, and John E.. Prussing. *Optimal Control with Aerospace Applications*. Springer, 2014.

VITA

VITA

Shreyas Vathul was born in Bangalore, India on August 30, 1989. He completed his B.Tech. (2007 - 2011) from National Institute of Technology Karnataka (NITK) in Mechanical Engineering. He completed his M.S from Wright State University (dayton, Ohio) in 2012 while working with the Flow Simulation Research Group. At Purdue University's School of Aeronautics and Astronautics, he worked as a Research Assistant in the Systems-of-Systems Laboratory (SoS Lab.) headed by his thesis advisor, Daniel A. DeLaurentis on topics such as aircraft design and optimization, air-traffic management and algorithm development.